PHYSICAL DESIGN AUTOMATION FOR LARGE SCALE FIELD PROGRAMMABLE ANALOG ARRAYS

A Dissertation
Presented to
The Academic Faculty

By

Ismail Faik Baskaya

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
in
Electrical and Computer Engineering

School of Electrical and Computer Engineering
Georgia Institute of Technology
December 2009

PHYSICAL DESIGN AUTOMATION FOR LARGE SCALE FIELD PROGRAMMABLE ANALOG ARRAYS

Approved by:

Dr. David V. Anderson, Advisor
Assoc. Professor, School of ECE
Georgia Institute of Technology

Dr. Abhijit Chatterjee
Professor, School of ECE
Georgia Institute of Technology

Dr. Sung Kyu Lim, Advisor
Assoc. Professor, School of ECE
Georgia Institute of Technology

Dr. Aaron Lanterman
Assoc. Professor, School of ECE
Georgia Institute of Technology

Dr. Paul Hasler
Assoc. Professor, School of ECE
Georgia Institute of Technology

Dr. Daniel Foty
President
Gilgamesh Associates LLC

Date Approved: July 31, 2009

DEDICATION

To my father, mother, and sister

for their love that will last forever.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, David V. Anderson for his continued understanding, support, and encouragement to overcome all the difficulties encountered in this pursuit. I have learned much from him; not only about our research, but also on many aspects of life. I hope I can be a vessel to pass on some of his light to other people. I would also like to thank my co-advisor, Sung Kyu Lim for his support, guidance and discipline, all of which were required for continued progress toward the end goal.

My thanks also go to my respected committee members Paul Hasler, Abhijit Chatterjee, Aaron Lanterman, and Daniel Foty for their valuable time and feedback on my work. I want to thank Paul Hasler additionally for his partial financial support that made completion of this project possible.

The contribution of my other valued teachers in bringing me to this point can never be forgotten; among them, my special thanks go to Omer Cerid, Gunhan Dundar, Gina Minyard, Gyuszi Suto, and Husniye Midik.

This was a long road with many hardships, but also has given me the unique opportunity to meet many people that I would not know otherwise. I would not succeed without the warm friendship, selfless support, and valuable feedback from my dear labmates and friends. Thank you all Walter, Jungwon, Tira, Ken, Brian, Venkatesh, Sourabh, Sunil, Sanmati, Shyam, Tyson, Hawjing (Mike), Bo, Nikolaos, Daniel, James, Taehyung, Christal, Arindam, Stephen, Jordan, Craig, Csaba, Shubha, Degs, Aurele, Guillermo, Abhishek, Woosuk, Devangi, Jorge, Mongkol, Young Joon, Dae Hyun, Michael, Mohit, Xin, Sermet, Utku, Ozgur, Pinar, Lakshmi, Tushar, Menderes, Zafer, Erhan, Cenk, Ebru, Arkadas, Musemma, Seda, Ozkan and many other valuable friends that deserved to be here.

Last, but certainly not the least, I would like to thank Janet Arnold and Pam Halverson for their great administrative support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Field-programmable analog arrays (FPAA) are integrated circuits with a collection of analog building blocks connected through a wire and switch fabric to achieve reconfigurability similar to the FPGAs of the digital domain. Like FPGAs, FPAAs can help reduce the time and money costs of the integrated circuit design cycle and make analog design much easier. In recent years, several types of FPAAs have been developed. Among these, FPAAs that use floating-gate transistors as programming elements have shown great potential in scalability because of the simplicity they provide in configuring the chip. Existing tools for programming FPAAs tend to be device specific and aimed at specific tasks such as filter design. To move FPAAs to the next step, more powerful and generic placement and routing tools are necessary.

This thesis presents a placement and routing tool for large-scale floating-gate-based FPAAs. A topology independent routing resource graph (RRG) was used to model the FPAA routing topology, which enables generic description of any FPAA architecture with arbitrary connectivity including possible FPGA support in the future as well. So far, different FPAA architectures have been specified and routed successfully. The tool is already in use in classes and workshops for analog circuit and system design. Efficient ways to describe circuits and user constraints were developed to allow easy integration with other tools. Analog circuit performance was optimized by taking into account the routing parasitic effects on interconnects under various device-related constraints. Parasitic modeling allows simulation and evaluation of circuits routed on FPAA. Finally, a methodology was developed to explore the optimum architecture for a set of circuit classes by evaluating the efficiency of different architectures for each circuit class.

CHAPTER 1

INTRODUCTION

We are witnessing an age where consumer electronics products are becoming outdated in months, not years. In the fast pace of today's electronics design world, time to market has become more crucial than ever. Companies that can effectively use all available resources to cut the design cycle are getting an edge against their competitors. Electronics design processes, which are inherently iterative, must avoid the costly failures whether incurred as money or time loss. Meanwhile, IC manufacturing technology continues to improve, but the very high end processes are often accessible by only the big players. Application Specific Integrated Circuits (ASIC) also require large scale production to be feasible. Considering these factors, using reconfigurable arrays in the design process emerged as an alternative to reduce time and money costs. Figure 1.1 demonstrates the potential to save money, time, and energy by inserting the reconfigurable arrays into the design process. Since their introduction into the market in 1985 with Xilinx XC2064, Field-Programmable Gate Arrays (FPGA) have been attractive reconfigurable options for digital system design and testing. FPGAs not only give engineers the flexibility of trying their incomplete designs and bring them to perfection, but can also hit the market early enough with a smaller investment for a limited production allowing the small players into the game as well. Today, FPGAs continue to receive interest from both engineers who use them for product prototyping and CAD researches who try to find ways to get the best out of these versatile devices.

However, time to market and design costs are not the only factors that determine the success of the design process. As embedded computing becomes mainstream, a significant market has emerged for feature-rich signal processing devices that consume very little power. While digital processors and FPGAs can perform the desired functions, there are many cases where an analog design can offer the same functionality at a fraction of the

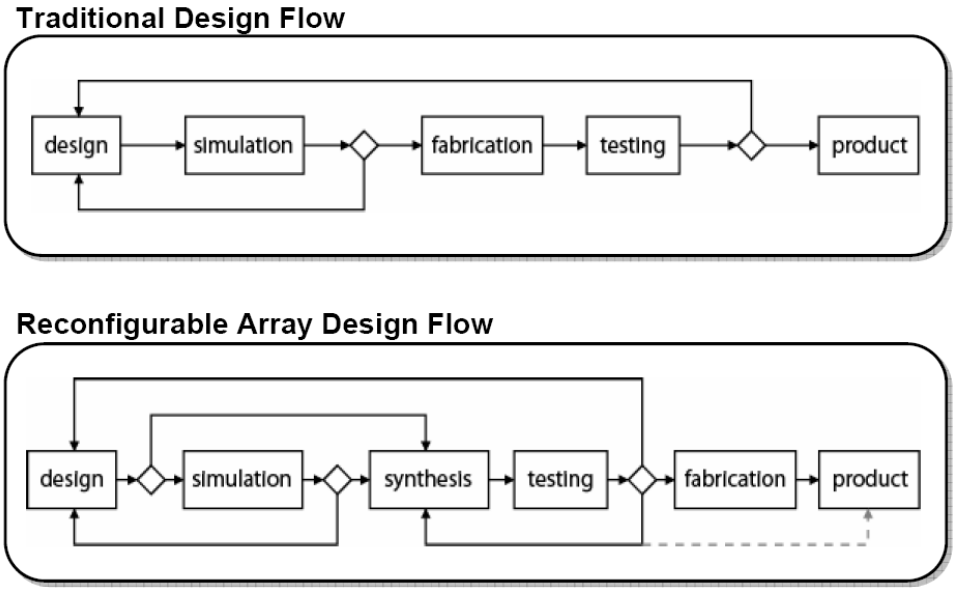


Figure 1.1. Traditional and reconfigurable array design flows.

power required for the digital solution [1]. For computations that don't require very high resolutions, an analog solution is less expensive than a digital solution. Figure 1.2 demonstrates the resources required for the same task of multiplying two signals with 16 distinct levels. A 4-bit digital multiplier requires 466 transistors, which is 66 times more than the number of transistors in a Gilbert multiplier that can perform as well for this resolution and maybe higher. In cases where a fast, approximate result is the real need rather than a very precise one, analog systems can replace digital systems with huge power and chip area savings.

However, analog design is expensive. One option that has become more viable in recent years is the use of Field-Programmable Analog Arrays (FPAA). FPAAs are analogous to FPGAs in that they allow designers to rapidly prototype circuits without having to fabricate new ICs. As FPAAs become more capable with their increased sizes and resources, the need for new CAD tools becomes more apparent. On the other hand, the methods that work for the physical design of digital circuits in FPGAs don't work very well for FPAAs. The criteria for a successful design are different and more complex. Signal integrity of an analog circuit is more difficult to maintain and can severely impact the functionality of the

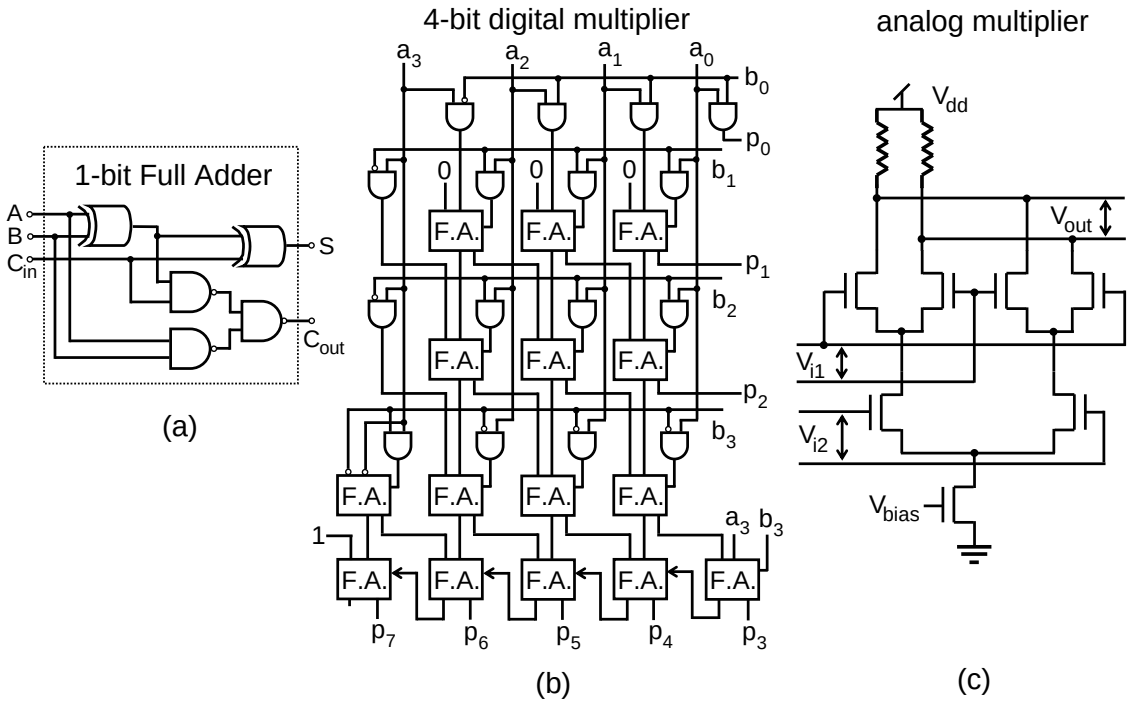


Figure 1.2. Analog solution vs digital solution for multiplying two signals at 4 bit resolution. (a) 1-bit Full Adder contains 24 transistors. (b) A 4-bit multiplier contains 466 transistors (15 Full Adders, 12 AND gates, and 5 inverters). (c) A Gilbert multiplier cell contains only 7 transistors and 2 resistors.

circuit; therefore, it is of higher priority than achieving the most compact design. Increased segmentation of wires may result in additional capacitances that can result in undesired circuit behavior after routing. Another challenge to be faced is the fact that parasitics not only deteriorate the performance as in digital circuits but may also destroy the functionality completely, which requires monitoring the impact of parasitics on performance metrics during the synthesis steps. A placement and routing solution that satisfies all device and net constraints may not necessarily be a desired one; the performance often has to be optimized as well.

The primary goal of this research is to make large-scale floating-gate-based FPAA technology [2] more accessible and more practical by means of automated placement and routing tools. Existing tools do not consider signal integrity issues, and they have been developed to program specific FPAA architectures with a small number of CABs, which are dedicated to specific tasks such as filtering; therefore, they are not appropriate for the new

large-scale FPAAs, which have more flexibility and variety in the type of tasks that can be performed. Our work is targeted at a particular type of large-scale FPAA that uses analog floating-gate elements for programming both routing and configuration of the components; however, it can be extended to support other FPAA technologies as well. The details for this technology is discussed in Section 1.2, and a historical background of the problem will be presented in Section 1.3. Chapter 2 describes the CAD tools developed for automatic placement and routing of analog circuits on the target large-scale FPAAs. In Chapter 3, a model for simulating the impact of non-ideal interconnects is presented. In addition, a methodology for using this model in optimization is described. In Chapter 4, a methodology for FPAA architecture exploration and results from evaluation of these architectures is presented. Chapter 5 gives an insight on possible directions this research can lead to, and concludes this thesis.

1.1 FPGA vs FPAA Physical Design

Design automation, specifically in the phases consisting of clustering, placement, and routing, is often deeply influenced by the architecture and device model of the implementation medium one is working with. Although there have been general solutions proposed to handle a wide variety of architecture models, usually these design tools are not as efficient as techniques customized specifically for pre-defined architectures. This same idea applies to using FPGA design automation methods for solving FPAA applications. The similarities and the differences between FPGA and FPAA physical design flows can be viewed as depicted in Figure 1.3.

A comparison between FPGA and FPAA design decisions can be approached from three fronts: the actual algorithms that are used, the metrics used in the algorithms, and the architectures used in the design. Some of the traditional FPGA clustering algorithms include multi-level clustering based on delay/area minimization to reduce the number of

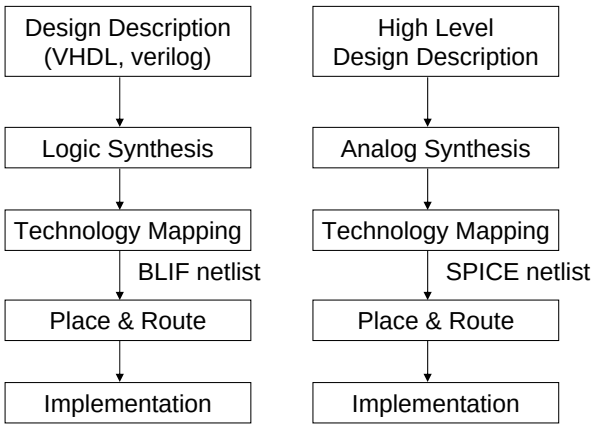


Figure 1.3. Steps in the physical design flow of FPGA and FPAA.

repeatable blocks. These algorithms are highly influenced by the device model one is dealing with and will not work without major changes for the FPAA hardware implementation. The device and interconnect constraints in the floating-gate-based FPAA make traditional FPGA algorithms unapplicable. During the placement phase, FPGA architectures often use simulated annealing or genetic (evolutionary) algorithms to achieve an optimal solution. Although similar ideas can be applied to FPAA implementations, the cost functions in these algorithms vary. Typically, higher priority is placed on reducing the wire length, net density, and overall delay in placement and routing for FPGA or other digital systems. On the other hand, FPAA and analog architectures need to focus on reducing the loading effects of routing parasitics and distortion of the signals.

The device architecture plays a major role in the physical synthesis decisions that are made. For instance, FPGAs traditionally have a very simple interconnect network architecture, where the vertical and horizontal channels have the same number of routing tracks and propagate the same signal down a pipeline. Each horizontal and vertical wire is segmented and is shared among several interconnects. This varies quite significantly from most FPAA, where the interconnect wires are usually not segmented and a single interconnect occupies the entire vertical/horizontal highway. The basic logic element used in FPGAs is called configurable logic block (CLB), and is a combination of programmable lookup

tables with universal functionality and flip-flops in general; whereas a configurable analog block (CAB) in a FPAA consists of many more various programmable analog components with distinct functionality. In addition, there are no sequential elements in a FPAA. Thus, behavioral as well as physical synthesis needs entirely different approaches to handle new cost functions under new types of device constraints. Thus, it is fairly difficult to directly associate automation algorithms between FPGAs and FPAAs.

1.2 Floating-gate Based FPAA

A floating-gate element is a polysilicon layer that has no contacts to other layers; this polysilicon layer can be the gate of a MOSFET and can be capacitively connected to other layers. Charge on the floating gate is stored permanently, providing a long-term memory because it is completely surrounded by a high-quality insulator. The charge on a floating gate can modulate a MOSFET's channel current; therefore, the floating gate is not only a memory, but also can be an integral part of a computation. The charge on the floating gate is modified through a combination of hot-electron injection and electron tunneling [3, 4, 5]. The small size and scalability make these approaches ideal for integration with classical analog techniques (e.g., voltage references, filters, ADCs, or DACs) as well as larger-scale analog signal processing techniques (e.g., compression or classification for audio or image signal processing).

Floating-gate transistors function in two ways in the FPAA: as switches that connect device pins or as configuration devices for analog components in the CABs. Using floating-gate transistors as switches have advantages over using a standard PFET or a transmission gate. With floating-gate transistors, the resistance of the switch can be controlled by the amount of injection, and a high aspect ratio is not required to obtain lower resistance values. Floating-gate switches also do not require complementary control signals as in transmission gates. Moreover, their resistance exhibits more linearity and consistency over the operating voltage ranges. Figure 1.4 gives a comparison between different switch elements [6].

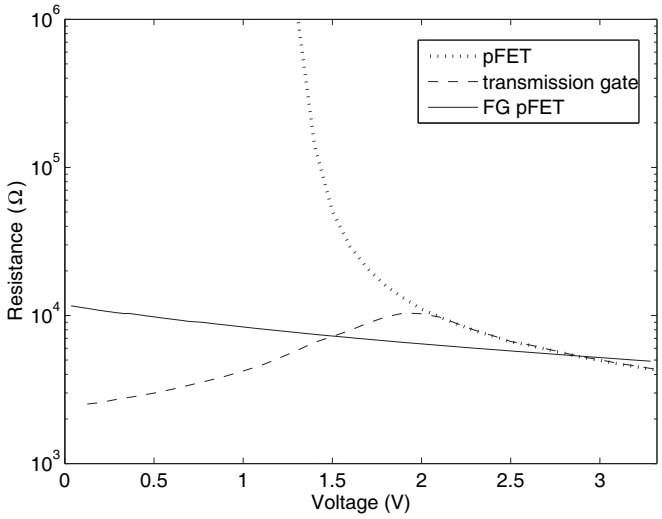


Figure 1.4. An empirical comparison of the on-resistance of three switch elements: a PFET, a transmission gate, and a floating-gate PFET where all of the pFETs are 3.6 μm wide by 1.2 μm long, operated with a supply of 3.3 V.

When floating-gate devices are used as just plain switches, they would look like any other FPAA, except for the small size and smaller level of parasitic resistance (R) and capacitance (C), which allows us to build larger arrays. But floating-gate switches can be used more than just as switches; we can actually use them in place of transistors with fixed gate voltages as well. They can act as anything from resistors and spreading elements to current sources and bias elements.

1.3 Existing Work on FPAA Research

The first configurable analog systems appeared with the use of analog computers in the 1960s. An analog system that can be configured using current signals and varying length of coaxial cables was used for preprocessing a radiation detector signal [7]. Another computing system that had the flexibility to configure the system topology and component parameters was intended for power system simulation for transformers, transmission lines, and valves [8]. An analog signal processor (ASP) that emulates z-domain filters by chopping the input signal and reconstructing it by means of a digitally controlled transmission gate [9] was developed in 1987.

At the end of 1980s and early 1990s, neural networks and related applications began to dominate the research on analog configurable systems [10, 11, 12, 13, 14]. Around the same time, introduction of cellular neural networks [15, 16] attracted attention to the configuration of circuits using only local interconnects between the neighboring computational units. A field-programmable IC using CMOS transmission gates for configuring the connections between various analog components targeted analog neural network synthesis and testing [17]. This architecture mostly suffered from the parasitic capacitances of the CMOS switches inserted into the signal path, and the routing wires; and could not store the circuit operating parameters. A programmable vector matrix multiplier targeting multi-layer neural network implementations takes advantage of adding the current-mode signals for free when they are connected together at a node [18].

There has been a transition from the earlier analog configurable systems to FPAAs after adopting a hierarchical organization of the architectures by collecting the computational analog elements into CABs. A low-power FPAA utilizing MOS subthreshold techniques introduced hierarchical interconnect architectures to FPAA design by collecting a set of analog components in CABs [19, 20]. This architecture was capable of storing circuit parameters in multi-valued memories as well. These research efforts produced further results as a MOS transconductor-based FPAA [21, 22] enabled by the programmable resistors used in the reconfigurable interconnect circuitry. These programmable resistors are essentially matched MOS transistors that are cross-coupled to eliminate the non-linear components in differential signals [23]. A current mode FPAA was implemented in bipolar technology, operating on the order of 100 MHz at the cost of a more expensive process technology and higher power consumption [24, 25]. Other current mode techniques were used in FPAAs based on current conveyors [26], and a folded cascode integrator [27]. There were efforts to build discrete-time FPAA architectures using switched-capacitor circuits as the configuration elements to operate in the voltage mode [28, 29, 30, 31], or switched current techniques to operate in the current mode [32] as well. FPAAs based on the switched-capacitor

technology are less sensitive to the effects of routing compared to their continuous-time counterparts; however, their achievable bandwidths are strictly limited by the switching frequency, making them an option only at the lower end of the frequency spectrum. The switched-capacitor FPAA technology developed by Bratt et. al. [28] was later commercialized by Motorola. Anadigm [33] is a leading commercial FPAA vendor distributing this product at present, and is currently supplying FPAAs with 4 (2×2) simple switch-capacitor based CABs. Later extensions made attempts to integrate both FPAA and FPGA components into field-programmable mixed-signal arrays (FPMAs) [34, 35, 36]. A survey paper [37] reviews in detail the early works mentioned above in the area of field programmable analog and mixed-signal circuits.

Several other FPAA architectures have been reported more recently. The interconnect structure in the field-programmable transistor array (FPTA) intended for evolvable hardware lets the user reconfigure the IC at the finest level of granularity; that is, the pins of each transistor can be individually accessed via the programming switches [38]. This strategy results in significant degradation of the target circuit performance because of the increased routing parasitic effects. A high-performance radiation-hard FPAA that uses anti-fuse technology has been proposed to reduce the parasitic effect and improve signal integrity [39]. However, this architecture is one-time programmable only, so can not be reconfigured for a different application. An architecture that increases the number of locally accessible CABs by tiling them in a hexagonal array eliminates the need for switches [40] utilizing output transconductance amplifier (OTA) gains for both computation and connection. However, this architecture contains only capacitors in addition to the OTAs, limiting the range of supported applications. There are also several works that focus on FPAA designs for testing [41, 42].

Currently several CAD efforts for FPAAs exist in the literature, including behavioral synthesis [43, 44], technology mapping [45, 46], and place-and-route [43, 47, 48, 46]. However, these works focus only on small-scale, switch-capacitor-based designs. On the

other hand, the floating-gate-based FPAAs in this study contain up to 84 (6×14) complex CABs at present, and still increasing; therefore, a more scalable approach is necessary to handle the complexity. In addition, the device and interconnect constraints in the floating-gate-based FPAAs are radically different from the switch capacitor-based FPAA. Significant work has been done on logical and physical synthesis for LUT (lookup table)-based FPGAs during last 20 years [49, 50, 51, 52, 53, 54, 55, 56, 57, 58]. As discussed earlier in Section 1.1, FPAAs have to be approached differently from the FPGAs because of the increased sensitivity of the target analog applications to the routing parasitics. Still, potentially useful approaches may be found among the placement and routing algorithms for FPGAs that may also be utilized for the FPAAs taking the analog design requirements into account.

1.4 Contribution

The major contributions of this work can be summarized as following:

- A placement and routing tool for large scale FPAA was developed. This tool is already in use by researchers and students for analog circuit and system design as well as FPAA architecture prototyping purposes.

- A generic FPAA architecture description interface enabled by a topology independent routing resource graph (RRG) was developed. With this interface, it is possible to describe a wide range of FPAA architectures with arbitrary connectivity, including possible FPGA and FPMA support in the future. So far, different FPAA architectures have been specified and routed successfully.

- A robust and efficient way of specifying input netlists along with FPAA specific constraints was developed.

- A parasitic extractor to simulate and evaluate the impact of the FPAA routing effects was developed. User can decide the accuracy and complexity of the extracted circuits

by providing different level SPICE models for CAB components, or by deciding the highest frequency of interest in simulation.

- Analog circuit performance was optimized by taking into account the routing parasitic effects on interconnects under various device-related constraints.

- A methodology was developed to explore the optimum architecture for a set of circuit classes by evaluating the efficiency of different architectures for each circuit class. This will be especially useful to help hardware designers in making strategic decisions for their prototype ICs.

CHAPTER 2

FPAA PLACEMENT AND ROUTING

Early FPAAs were just a collection of a few operational amplifiers and capacitors optimized for building filter applications [37]. For such small architectures with simple interconnect topology and small-sized circuits to program on them, manual placement and routing could be sufficient. The tools developed for these FPAA systems, such as Anadigm Designer[33], consisted of a schematic capture interface for the circuit entry that allows user to select and connect together circuit elements from the available analog blocks to perform desired functions. The complexity of such devices did not require sophisticated placement and routing algorithms that are often designed for larger scale circuits.

As FPAAs grow in size, several factors affect their usability. It is now possible to map larger circuits to the FPAAs because of their increased resources; however, this can be harder and more tedious to do without the aid of an automated tool. Larger systems also benefit from abstraction for ease of design, analysis, and description. Average user of the FPAA cannot be expected to know the details of the topology to be able to know where to place each circuit component, pick the wires to connect them, and generate a data stream. Architectural details should be hidden from the engineers whose real objective is to implement and test their conceptual designs without additional overhead. Furthermore, there is more than the number of components to consider for the circuits to be implemented on the FPAA. The limited number of wires and switches that constitute the routing resources of a small-scale FPAA had less impact on the circuit performance. As architectures grow, so does the number of interconnects and the parasitic impedances. They have to be accounted for more carefully to avoid destroying the functionality of the circuit. It is a very difficult task to account for the interconnect non-idealities as well when manually routing circuits. A tool must replace the human for tasks human cannot do efficiently. Placement and routing tools allow moving designs from one platform to another without having to manually

create new mapping tables if designed to support different architectures. This is especially useful in the availability of multiple FPAA architectures to choose from. Tools are also needed for architectures to improve. Developers of the FPAA must rigourously test their chips with circuits that continuously evolve and grow in size. A placement and routing tool will also evolve in parallel in addition to continued support for the improved architectures.

2.1 The scope of the tasks

Physical design for FPAA consists of several steps as depicted in Figure 1.3. This work focuses only on the placement and routing stages of the physical design for FPAA. Behavioral synthesis from high level specifications that is analogous to the logic synthesis for digital circuits is not within the scope of this study. Input circuits for the placer must be constructed using only the components that are available in the target FPAA. Therefore, the output of technology mapping is also assumed to be readily available. The placer takes the technology mapped circuit descriptions in the form of SPICE netlists, and finds an available circuit element or block on the FPAA CABs for each component of the implemented circuit. The router takes the output of the placer and finds switch coordinates that establish the necessary connections via routing wires between the pins of placed components.

One exception to starting from technology mapped circuits is the optional assignment of capacitance values from a signal net to ground. In this case, the objective is to achieve the specified capacitance value through the wires and discrete components if necessary rather than using the FPAA CAB components alone. This algorithm will be explained in more detail in Section 3.4. Another exception is the use of switch elements (SWE) in circuit descriptions. These circuit elements don't correspond to any component in FPAA CABs, but are emulated using the routing fabric. The algorithm and the conditions for using this element will be explained in more detail in Section 2.4.2.

In addition, simulation models for routed circuits are generated using a parasitic extractor. This is useful as a quick verification of the circuit performance after placement

and routing. The methods and results for parasitic extraction in different generations of the tools are explained in detail in Sections 3.1 and 3.3.

2.2 Input Circuit Interface

In digital circuits, each signal can be driven by the output of only a single gate. This condition prohibits the connection of two outputs, which would lead to ambiguous logic levels, even potential circuit damage when powered. Popular netlist formats such as BLIF reflects this limitation well [59]. When analog circuits are concerned, it becomes vague where the output of an element type can be, unless it's a high level circuit block with well-defined input and output ports. For basic elements, such as a FET, the assignment of input and output ports may vary depending on how this element is connected to the rest of the circuit. Furthermore, it is not uncommon to observe the outputs of multiple elements connected together. As an example, an inverter, which is a basic element from a digital perspective, should not be connected to another inverter at their outputs. The same inverter when viewed from an analog perspective, consists of a PFET and NFET as basic elements, both of which have their outputs connected to each other to perform the desired function. This makes the use of single gate-driven netlist formats which work very well for describing digital circuits inconvenient for analog circuit description. To reflect this nature of the analog circuits, SPICE netlist format is found to be very suitable. SPICE is a widely used EDA tool and is already familiar to most analog designers [60].

Input netlists used by the FPAA placement and routing tools are fully compatible with SPICE format; therefore, they can also be used for simulation in addition to circuit description. To enter placement and routing tool specific commands that are not available in SPICE while preventing SPICE from unrecognized keywords, a special text sequence that can be treated as a comment line by SPICE is used: "$* >>$". If just "$*$" is used, the line is regarded as a comment line by the placement and routing tool as well. The lowest level netlist entry allowed by both RASPER and GRASPER is a subcircuit that corresponds to

```
X1 Vin Vx Vx OTA PARAMS: Ib=100n
X2 Vout Vx Vdd PF
X3 Vout Vbias 0 NF
.....
.subckt NF d g s
M1 d g s 0 NFET W=... L=...
.ends
.subckt PF d g s
M1 d g s Vdd PFET W=... L=...
.ends
.subckt OTA p n out PARAMS: Ib=...
...
.ends

*>> project work
*>> pin io_lt 0 net Vin
*>> pin io_lt 1 net Vout
...
```

Figure 2.1. Sample SPICE netlist entry for FPAA placement and routing tool.

a component in FPAA. Therefore, the lowest hierarchy elements of the input netlist must be instantiations of subcircuits that contain information regarding the corresponding CAB component including simple analog elements such as capacitors and transistors. A sample SPICE netlist that can be used as an input to the placement and routing tool is presented in Figure 2.1. For additional details on the input circuit file format, refer to Appendix A.

2.2.1 Graph-based Internal Representation of Circuits

Each circuit element from the SPICE netlist is stored as a graph vertex (cell) and each signal node is stored as a hyper-edge (net). A net can be connected to the same cell from different terminals of the element; each connection point is modeled as a pin of the cell. The graph element equivalents of the circuit elements are shown in Figure 2.2a. Figure 2.2b depicts the equivalent graph representation of the circuit described by the netlist in 2.1. Since input and output terminals are not determined, signal flow direction is not clearly defined; therefore, representing the circuit with an undirected graph is preferred to using a directed graph.

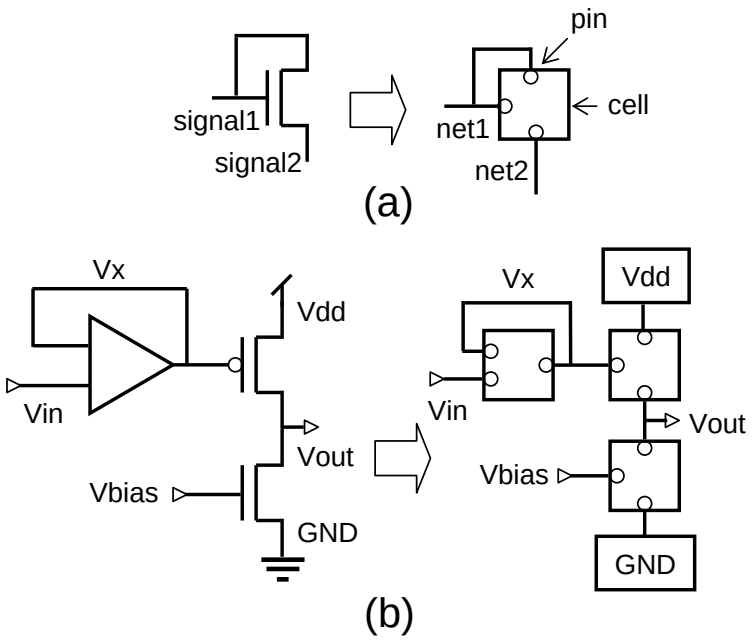


Figure 2.2. Graph representations of (a) circuit elements, and (b) a complete circuit with input and output pins.

Once a circuit is captured as a graph, the objective is to map each graph cell (i.e., circuit component) to an actual component on the target FPAA architecture and determine the switches that define a path between pins that need to be connected. Figure 2.3 illustrates an analog circuit mapping onto the actual FPAA.

2.3 Target Architectures

During the course of this project, different versions of tools emerged to reflect the changes in supported architectures as well as the features desired. These will be referred to as RASPER and GRASPER to distinguish between the capabilities and supported architectures between each tool. RASP2.5 and RASP2.7 architectures are built using CABs similar to RASP1.0, which is the first member of the RASP family [61]. RASPER was developed to support placement and routing of these architectures. When RASP2.8 was introduced later, the modified interconnect structure rendered RASPER obsolete, and GRASPER was developed to map circuits to this architecture.

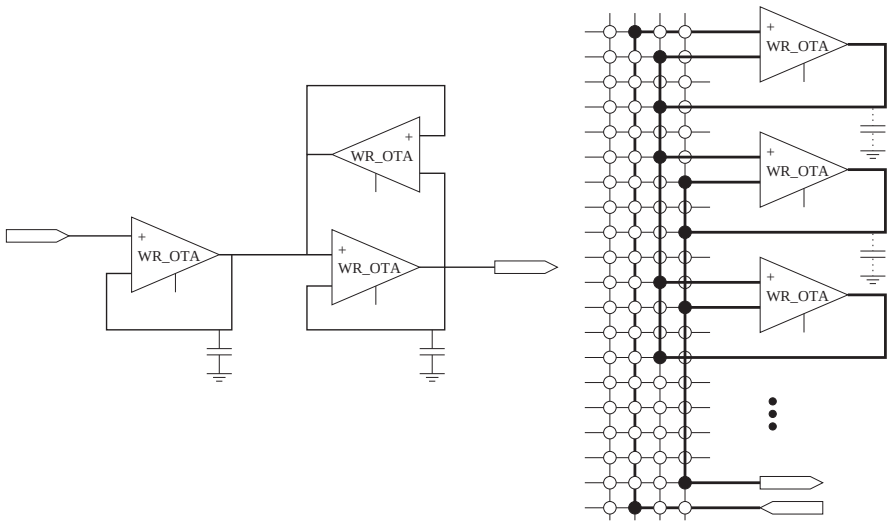


Figure 2.3. An analog circuit and its mapping to the target FPAA architecture.

2.3.1 RASP2.5 and RASP2.7

RASP 2.5(7) has 7(9) rows and 8 columns for an array of CABs that fall into two types: general purpose CABs, which contain a selection of basic components, and VMM CABs, which have a special purpose Vector Matrix Multiplier (VMM) circuit block in addition to the components of a general purpose CAB. Both architectures have been built using two-poly four-metal CMOS process with 0.35 μm minimum feature size.

The routing resources in RASP2.5 and RASP2.7 are shown in Figure 2.5. There exist three kinds of routing switch boxes: local, vertical, and horizontal crossbar. The columns in the local crossbar (named *vertical local wire* or *vlwire*) are used to establish connection among components in the same CAB. The vertical crossbar is used to connect components from CABs in the same column. Last, the horizontal crossbar is used to connect components from CABs in different columns. The routing switches in local, vertical, and horizontal crossbars are respectively called local, vertical, and horizontal switches. Each column in the vertical crossbar (named *vertical global wire* or *vgwire*) extends all the way from top to bottom, while each row in the horizontal crossbar (named *horizontal global wire* or *hgwire*) extends all the way from left to right.

Each row or column in these crossbars can only be used by a single interconnect. In

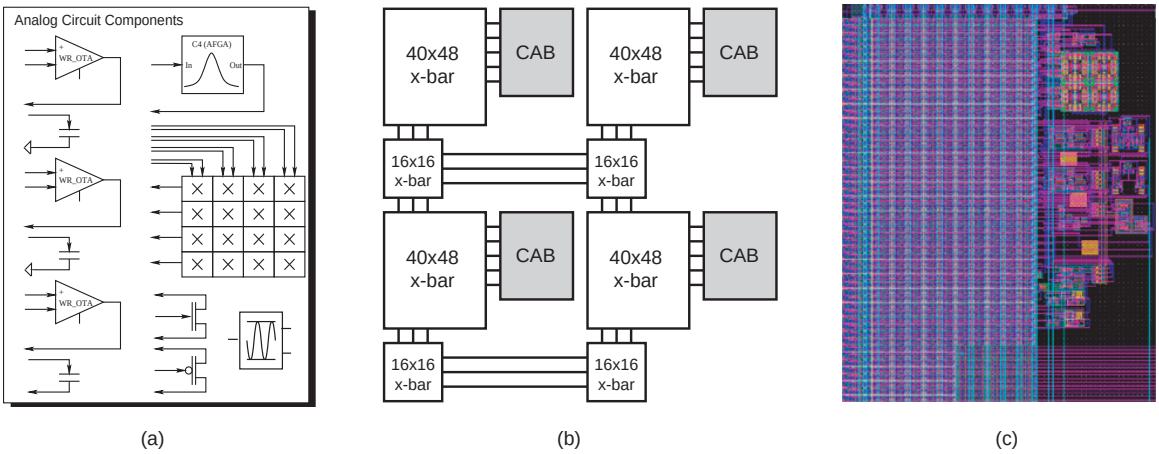


Figure 2.4. (a) Configurable Analog Block (CAB) for a FPAA based on floating-gate devices, where each CAB contains a four-by-four matrix multiplier, three wide-range operational transconductance amplifiers (OTAs), three fixed–value capacitors, a capacitively coupled current conveyor (C^4), a signal-by-signal multiplier, one PFET, and one NFET. (b) Overall block diagram for a large-scale FPAA. The switching interconnects are fully connectable crossbar networks built using floating-gate transistors. (c) Layout of a single CAB and its crossbar switch circuit. The switches are to the left of the actual components and dominate total area use by about 2/3 share.

addition, once a net occupies a row (or column), the sum of all routing switch capacitances in this row (or column) contribute to the parasitics of the interconnect. Each switch contributes a parasitic diode capacitance from the drain of the switch to ground regardless of the switch being on or off. This is the depletion capacitance, and the worst-case (highest) value of this capacitance is when the switch is off [62]. Since there is a number of switch capacitances in parallel on a line, the total capacitance of the line is assumed to be the sum of all these off capacitances. The difference that comes from the single on-switch on the line is negligible compared to the total line capacitance. When a switch is turned on, there will also be a resistance in series on the order of 10 kΩ [63].

The following three kinds of interconnects are possible, depending on the placement of components (see Figure 2.5):

- intra-CAB wires (type 1): these wires connect components in the same CAB using the switches in the local crossbar. We model the resistance of local switches that are turned on (= R). The capacitive component of the wire includes all local switches in the entire row (= C_w) and column (= C_l) occupied by the wire. The parasitics of

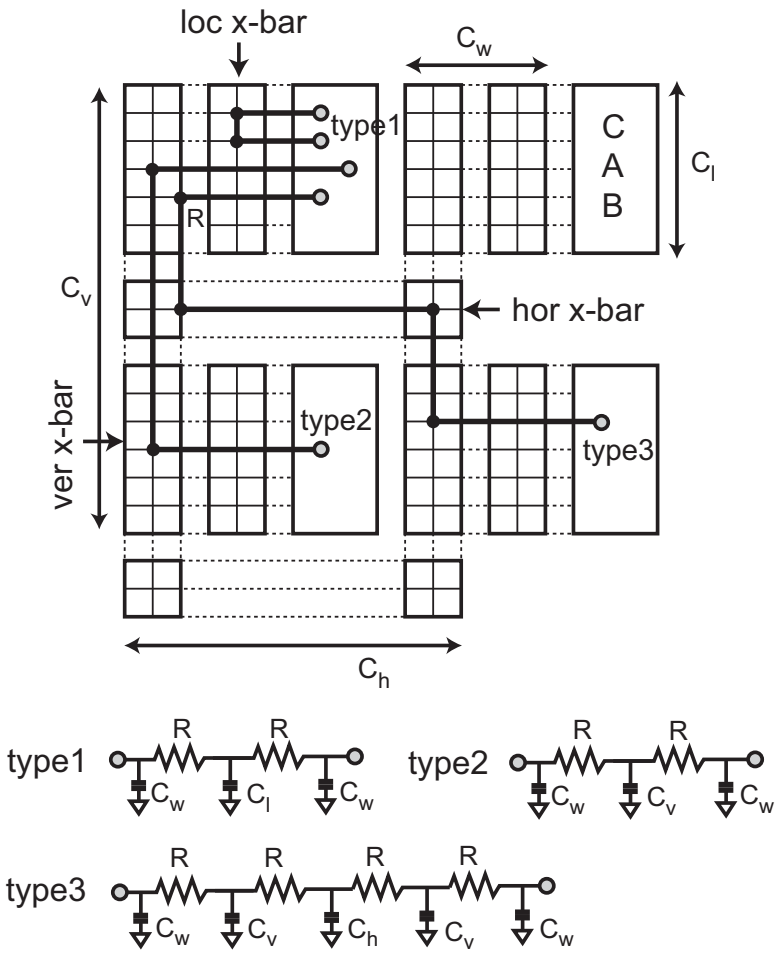


Figure 2.5. Routing resource for a 2×2 FPAA that consists of local, vertical, and horizontal crossbars. Three types of interconnects (1: intra-CAB, 2: inter-CAB/intra-column, 3: inter-column) are also shown.

these wires are minimal.

- inter-CAB/intra-column wires (type 2): these wires connect components from different CABs located in the same column. We model the resistance of vertical switches that are turned on (= R). The capacitive component of the wire includes all vertical switches in the entire row (= C_w) and column (= C_v) occupied by the wire. The parasitics of these wires are between those of type 1 and 3 wires.

- inter-column wires (type 3): these wires connect components from different CABs located in different columns. We model the resistance of vertical and horizontal switches that are turned on (= R). The capacitance of all vertical and horizontal

switches along the rows (= C_w and C_h) and columns (= C_v) occupied by the wires are modeled. The parasitics of these wires are maximal.

We assume that $C_w < C_v$, $C_w < C_h$, and $C_l < C_v$. In case the interconnect contains more than two components (= multi-pin net), each source-to-sink connection can be individually modeled. The type 1 wires are alternatively called *i-nets* in this proposal, whereas type 2 and 3 are called *x-nets*. Zero-value time constant analysis [64] on the type 1 interconnect yields

$$\omega \approx \frac{1}{R(2C_w + C_l)} \tag{2.1}$$

for the dominant pole only. The equation shows that each switch contributes a pole when added into the circuit path and has a negative impact on the bandwidth.

As a result of the exclusive usage of routing tracks (= rows and columns in the crossbars) by the interconnect, the number of available routing tracks imposes a strict design constraint. For example, the number of vertical tracks that have to be shared by all CABs in the same column is 10 in RASP2.5 and RASP2.7 architectures. The reason for this strict design constraint is because of the inverse relation between the number of routing switches in each track and bandwidth. An experiment was done earlier to explore the effect of routing configurations with different number of switch pairs and varying sizes of crossbar switch networks as depicted in Figure 2.6 [6]. Increased number of switch pairs makes the RC chains displayed in Figure 2.5 longer (i.e., increased resistance and capacitance), and the capacitance values at each node of these RC chains increase proportional to the crossbar switch network size. As a natural result, bandwidth drops as the crossbar switch network size or number of switch pairs increase as demonstrated in Figure 2.7. Architectural design space exploration can reveal good trade-off points between the number of routing switches and bandwidth degradation.

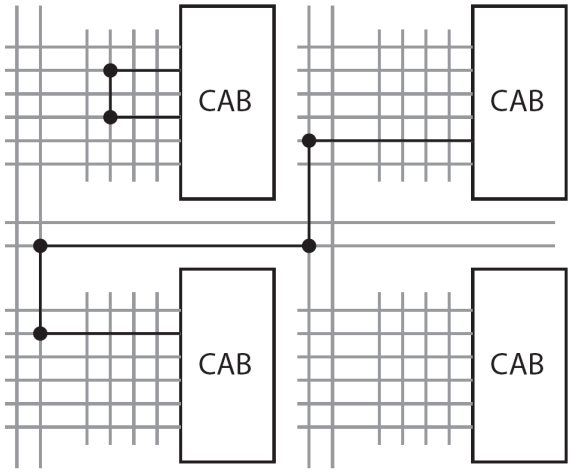


Figure 2.6. Typical routing scheme used in FPAA crossbar switches. Each CAB has a local crossbar network for routing devices within the cab. These local networks connect to global routing lines that run vertically between the CABs. Horizontal global routing lines connect the vertical routing lines across the chip.

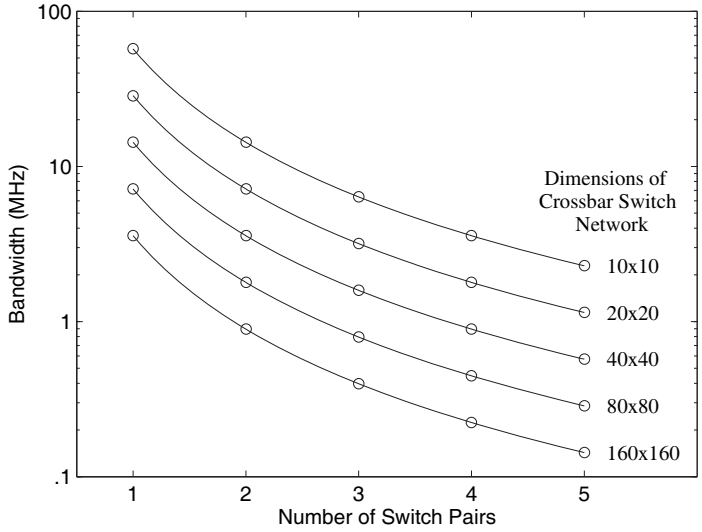


Figure 2.7. Switch network bandwidth for different network length and crossbar configurations.

2.3.2 RASP2.8

RASP2.8 [65] has an 8 row by 4 column array of two major CAB types. These CABs consists of analog building blocks or elements such as OTA, Floating-gate input OTA (FGOTA), buffer (a negative feedback self connected FGOTA), capacitor, field-effect transistors of n (NFET) and p (PFET) type diffusion, transmission gate (TGATE), multiple input translinear element (MITE), Gilbert multiplier, and a floating-gate input current mirror. Floating-gate transistors are used for some of these elements either as input stages to allow controlling the offsets resulting from device mismathches, or as bias elements that help configuring the operating point of the circuit blocks. The first major type contains all component types except current mirrors and multipliers; the second type contains current mirrors, multipliers, and FGOTA. RASP2.8 has been built using two-poly four-metal CMOS process with 0.35 μm minimum feature size. A die photo of the fabricated IC and the CAB types with internal compoenents are presented in Figure 2.8.

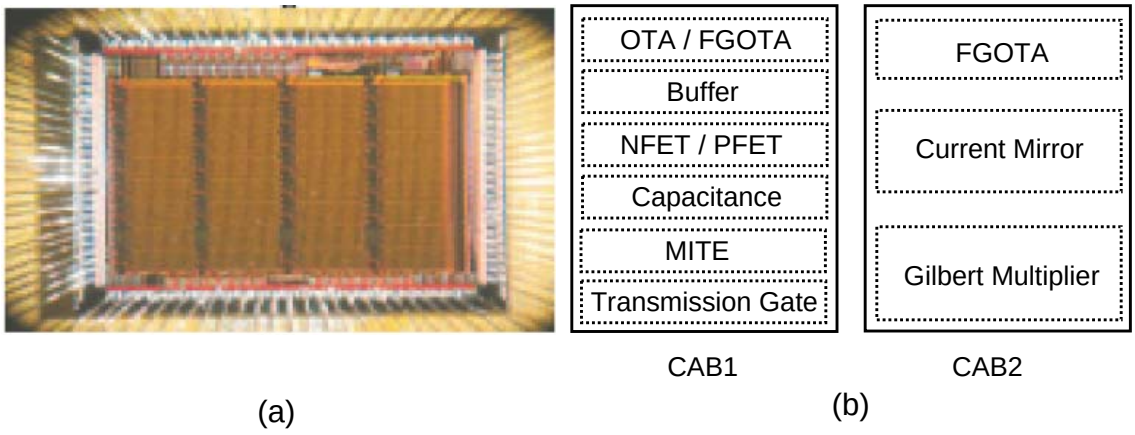


Figure 2.8. RASP2.8 architecture (a) Die photo of the fabricated IC. (b) CAB types with internal components.

RASP2.8 routing structure brings major changes compared to the previous architectures in the RASP family. In addition to the vertical local and global wires used in RASP2.7, there are also vertical and horizontal nearest neighbor wires in RASP2.8. By adding these wires, the mesh interconnect style that is widely used in FPGA interconnect topologies is also incorporated into RASP2.8. The most important change in the wire topology is the

existence of bridge switches between the consecutive wire segments on a line; whether they are local wires (spans one CAB row) or nearest neighbor wires (spans two CAB rows). In RASP2.5 and RASP2.7, wires on a line were completely isolated from each other, and all inter-CAB connections had to go through a single global wire to minimize the number of switches, highly limiting the inter-CAB routing options, and wasting a whole column even if just a small portion of that column is intended to be connected. RASP2.8 allows wire segmentation, which utilizes the routing area better, reducing the parasitic capacitance on each wire segment, and relaxing the strict design constraints on the global lines at the cost of additional switches to be added on the signal path.

RASP2.8 also has dedicated power lines for VDD and ground that span all columns vertically and can be accessed by any component pin without using additional wires or I/O pins. RASPER was designed based on the simpler RASP2.5 and RASP2.7 interconnect topologies, which provided very limited options for routing; effectively associating each placement solution with only one routing solution if it exists. The new wire types, wire segmentation levels and the possibility of connecting segmented wires through the bridge switches in RASP2.8 increase the routing configurations and the interconnect types greatly. With the increased routing possibilities in RASP2.8, RASPER can no longer support the interconnect model; therefore, GRASPER was created to map circuits to this architecture and possible future extensions. The routing structure with all wire types in RASP2.8 architecture is depicted in Figure 2.9.

2.4 RASP Placer and Router (RASPER)

RASPER is the initial attempt to solving problem [66]. An illustration of the architecture modeling for RASPER is shown in Figure 2.10. Since the available FPAA architectures at the time RASPER was developed were limited to RASP2.5 and RASP2.7, their requirements and limitations were of main concern in the development of the tool. Historically,

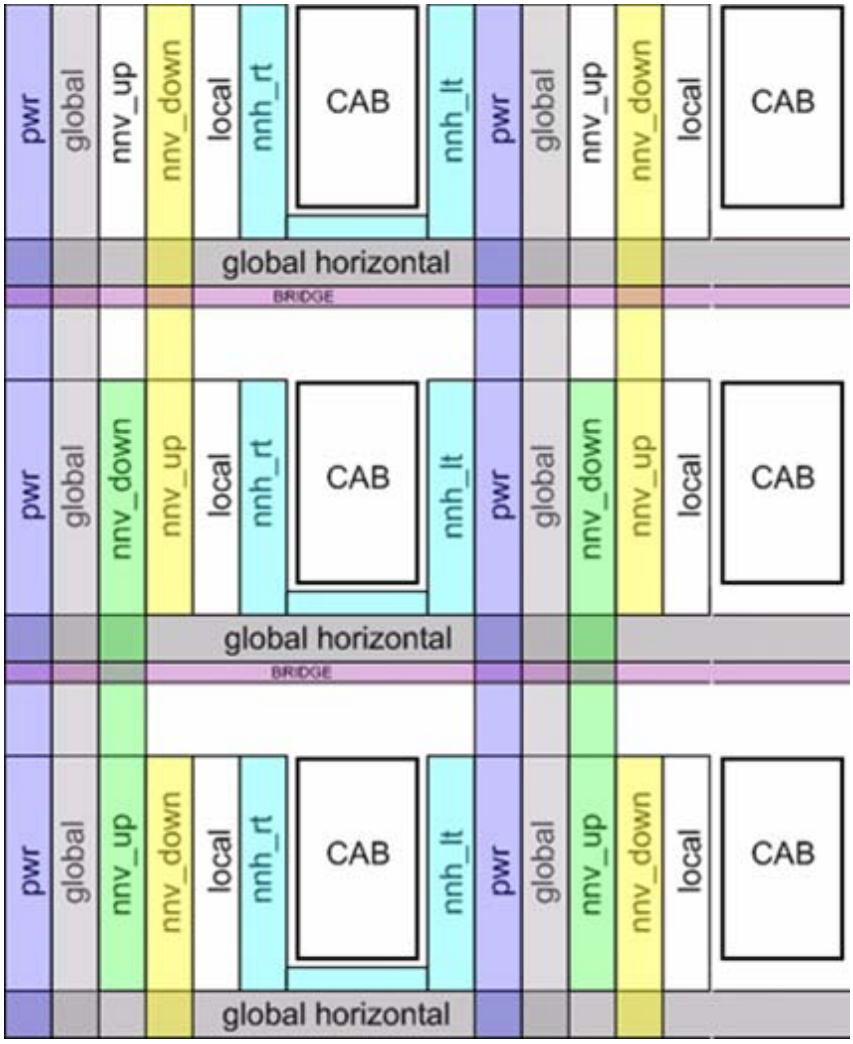


Figure 2.9. Routing structure of RASP2.8 architecture showing multi-level routing lines with different capacitances for improved bandwidth and connectivity.

RASPER's development can be divided into two phases: In the first phase, a classic approach was taken to develop a CAD tool with clustering, placement, and routing stages to map randomly generated circuits built using the available FPAA component types. In the second phase, the focus was on creating a CAD tool that takes the special requirements and limitations of analog circuits.

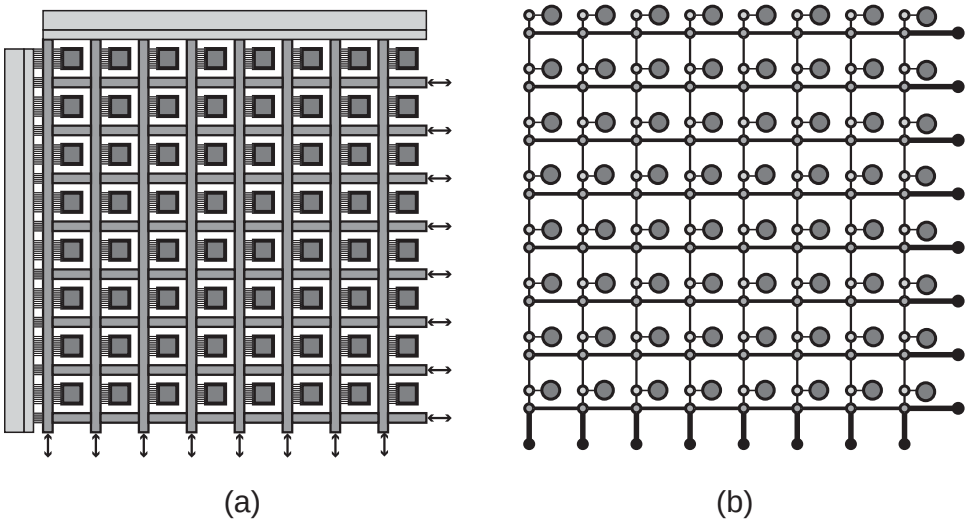


Figure 2.10. Illustration of graph-based FPAA modeling in RASPER. (a) 8X8 FPAA, (b) its graph-based representation, where big and small nodes respectively denote the CABs and routing switches.

2.4.1 Clustering, Placement and Routing in Early Development Phases of RASPER

The first step in RASPER is the FPAA clustering stage, which groups the analog circuit components together to form CAB clusters. The constructive CAB clustering algorithm used consists of two steps:(i) cell ordering for clustering priority, (ii) CAB ranking and selection. In a nutshell, cells are visited in a certain order and the best possible CAB to cluster each cell with is searched while monitoring various constraints. The cells are ordered according to the MHEC scheme, which is used in circuit partitioning for cut-size minimization that effectively increases the use of local wires for routing [67]. In this scheme nets are visited in ascending order of their sizes, where the size of a net is the number of cells it connects. Clustering smaller groups first fills the CAB slots more slowly and it is less likely to divide the cells from a single net into different CABs in the earlier stages of

clustering (i.e., when there are more slots available). In addition, it is easier to assign cells in smaller nets to the same CAB. The motivation is to break as few nets as possible by focusing on the smaller nets first, so that the total number of inter-cluster connections is naturally minimized. Cells of each net visited are ordered in a random order. The net size ties are again broken randomly.

Clustering stage is followed by the FPAA placement stage, which is the process of assigning each cluster to a physical CAB on FPAA. Placement stage consists of two parts; namely, constructive placement and refinement. In constructive placement, clusters are listed in descending order of their external connections (i.e., nets that have cells that are both inside and out of the cluster) while CAB columns are listed in ascending order of the wires that allow connection between different CABs. In RASP2.5 and RASP2.7 architectures, these wires correspond to the vertical global wires. The limitations on these vertical global wires that must be shared by all CABs are so severe that it is often not possible to find a valid placement solution with existing number of vertical global wires. To overcome this issue, the constraint for the number of available vertical global wires is temporarily relaxed, and then the resulting violations are removed during the refinement phase. Refinement uses simulated annealing (SA), which is a widely used stochastic optimization method. SA-based refinement starts with an initial placement solution and allows swapping the CABs associated with each cluster at every iteration. If the new placement solution leads to a better arrangement (reduced number of vertical global wire use in this case), the change is accepted. If the new arrangement has a higher cost than the previous placement solution, the change is accepted with a certain probability, which reduces as the refinement process goes, simulating the temperature drop in the actual annealing process [68].

The routing stage is the final and the most trivial stage that takes place after placement is complete. In this architecture, there is no difference between choosing any two vertical local wires of the same CAB from each other. The same condition also applies to choosing

between two vertical global wires of the same column. So, there exists only one possible route for every connection of placed cells if there is a feasible route. If all pins of a net belong to the cells placed in the same CAB, a local wire is assigned to that net; otherwise, a global wire is used. Routing is completed after determining the switches that connect the selected wires of each net to each other.

To test the performance of RASPER at this stage, a set of circuits were created. These circuits were not designed for realistic functions; they were just randomly connected components selected from the pool of available FPAA component types. Since functional evaluation was not possible for these circuits, performance of the placement and routing was evaluated based on the utilization of FPAA resources (CABs and wires) and the success in minimizing the use of switches.

2.4.2 Simultaneous Placement and Routing in RASPER

The scheme used in early versions of RASPER has little success with real circuits and the actual FPAA architectures. It is very difficult to satisfy the net constraints with the high demand on and the scarcity of the vertical global wires. Without tracking the availability of these wires, the clustering stage often gives outputs that are impossible for the placer to fit on the real FPAA unless vertical global wire count is increased, and the refinement stage can not always remove the violation of these constraints.

A solution to this problem is to associate each cluster with a physical CAB immediately after it is created. This effectively combines the tasks of the clustering and the placement stages. The main reason for using clustering is to reduce the number of components; therefore, the problem size for the placement step. Although by using a placement and routing software we expect to increase the size of the circuits that can be implemented on the FPAA, we can also assume that their sizes will never be comparable to that of the digital circuits. The additive natures of noise and parasitic effects make it impossible to catch up with the digital circuits in terms of component sizes; besides, the original intention for using analog circuits is to replace the much larger digital circuits that do the same task. So, incorporating

the task of clustering into the placement step is not expected to have a huge impact on the synthesis time when analog circuits are concerned.

Routing is also trivial with the available routing topology as mentioned in the previous section. A set of placed component pins that belong to the same net can establish a valid connection in only one way. In this sense, routing essentially can serve as a constraint for placement rather than a separate step, so it is logical to combine them as well. Therefore, RASPER was redesigned as a simultaneous placement and routing tool. The first placement and routing results with actual circuits and architecture are also obtained after these modifications. Some of the circuits that have been placed and routed using RASPER will be presented in Section 3.1.

Placement and routing for analog design differs from digital. In digital design, parasitics usually affect only the performance (speed) of the circuit. In analog design, they can have a major impact on the overall functionality as well. Therefore, the effects of the parasitics on the circuit performance are more crucial than in the digital case. What makes things worse is that incremental analysis methods are not well suited for analog circuits. The effects of the parasitics coming from placement and routing results have to be analyzed as a whole. To monitor the performance of placed and routed circuits, RASPER was further modified by adding parasitic extraction and post-routing simulation features. In addition, chip I/O pins can also be described with different parasitic effects on the wires to reflect their distinct effects on the circuit performance.

In the new version of RASPER, support for a new element type that is called as switch element (SWE) is added. This element type was possible by taking advantage of using the floating-gate switches in RASP2.5 and RASP2.7 architectures instead of the more traditional transmission-gate based or other switch technologies. As mentioned in Section 1.2, it is possible to program a floating-gate transistor in a way that gives more control over the voltage-current relationship of the switch. This allows utilization of the switch fabric as part of the circuit rather than dead weight. RASPER recognizes the entry of SWE in the

circuit netlist and rather than matching it to a CAB component, seeks switches for this element among all the routing switches. An example for how SWE can be used is illustrated in Figure 2.11.

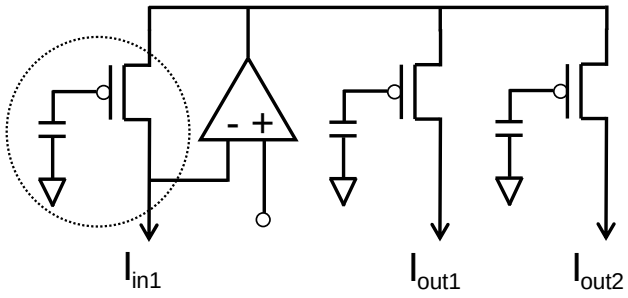


Figure 2.11. Illustration of switch elements (SWE) on a circuit. The encircled transistors are not mapped into the matching CAB components, but implemented as the routing switches instead.

After every cell is associated with a CAB component and wires are associated with nets and SWEs, an output file that contains the switch coordinates and associated current values is generated to be used for the final stage of implementing the circuit on the FPAA. This file is the final output of RASPER. There are three types of switches in the output file:

- Bias switches are used to set the bias currents for some FPAA components. These components are analog building blocks rather than a single circuit element, and require a configuration value for operating in the desired range. Setting the bias current of an OTA can be counted as an example. Floating-gate switches used for configuring these circuit blocks serve to reduce the demand on I/O pins of the IC. Bias switches need to be associated with a current value that determines the bias level of the related circuit block.

- SWE, as explained above, uses switches from the routing switch fabric. Since programming them to certain voltage-current relationships is desired, they are associated with current values that contain programming information.

- Routing switches establish the connections between component pins and other component pins or I/O pins. Among all switch types, only routing switches do not require

a current value, since the programming interface can automatically decide the appropriate gate charge they should be programmed to.

With all these modifications made in RASPER, implementing real circuits on actual devices became possible for the first time. As a result, more realistic circuits were designed to test placement and routing algorithms. Some of these circuits can be found in Section 3.2.1.

2.4.3 RASPER Placement and Routing Results

In the beginning of our work on physical synthesis for FPAAs, there was a lack of established standard benchmarks suitable for use with the available component set. Initially, there were attempts on generating random circuits to test RASPER; however, these circuits were highly unrealistic, thus neither would reflect the nature of actual analog circuits nor would be of any use in observing the impact of synthesis on circuit performance. Later, a smaller set of realistic analog circuits were designed to test the success of RASPER. Although replacing some circuit blocks with specialized components would enhance the overall circuit performance, certain component types of RASP2.7 were avoided since their functionality were not completely verified at the time of these experiments. Table 2.1 presents the resource usage of these analog circuits mapped to RASP2.7 using RASPER. One thing that can immediately be observed is that no matter how many components and nets the circuits have, the use of local routing is very limited. On the other hand, there is a very high pressure on the use of some resources. These results imply that RASP2.7 may not be very compatible with this set of circuits. It is quite possible that most circuits do not obey the component ratios of the available CABs in this architecture, thus the local routing resources were not be utilized well. Therefore, exploring and evaluating the fitness of architectures for different circuit classes is very important for better utilization of the devices. Detailed description and performances of these circuits will be presented in Section 3.1.

Table 2.1. Resource usage for several analog circuits mapped to RASP2.7 using RASPER.

circuit	component	net	CAB	hgwire	vgwire	vlwire	switch
bpf	5	5	1	0	3	2	39
vco	15	15	10	7	23	0	574
dac8bit	9	19	9	10	31	0	364
wta-3way	13	14	7	12	17	0	426

2.5 Generic Reconfigurable Array Specification and Programming Environment (GRASPER)

RASPER was developed to be somewhat specific to the interconnect topologies of RASP2.5 and RASP2.7 architectures. Having only two types of wires to connect to the component pins within each CAB enforces only one routing solution for any given placement arrangement of the components. RASP2.8 introduces the bridge wires that allow connections between the new segmented wire types of various spans in addition to the existing local wires. This change dramatically increases the routing possibilities, making it impossible to use the simultaneous placement and routing method in RASPER.

The data structure based on the CAB-wire interactions is also insufficient to model the diversified interconnect topology of RASP2.8. In RASPER, wires can have membership of either a column of CABs or just a single CAB, which can not address the mesh style wires that span two CABs along vertical and horizontal orientations. In RASPER, switches are not represented explicitly, but are extracted from a set of wires that are occupied by a net after placement and routing. With the increased number of wire connection possibilities, it is not possible to extract a unique set of switches from a given set of wires.

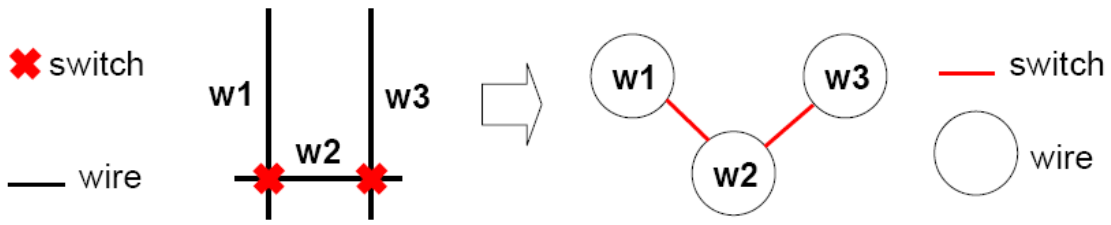


Figure 2.12. Wires-switch relationship on a reconfigurable architecture. GRASPER uses a model where wires are represented as vertices and switches are represent as edges.

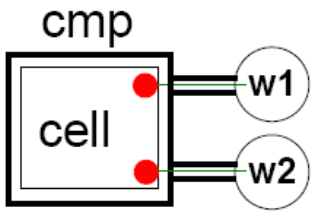


Figure 2.13. Component pin-wire connection modeling in GRASPER. This model allows instant determination of terminal wires (source/sink) for routing immediately following placement.

GRASPER uses a different RRG model that depends on capturing the wire-switch relationship as a vertex-edge pair in a simple undirected graph as illustrated in Figure 2.12. In this undirected simple graph model, every edge must connect always two distinct vertices, and a vertex connects to another vertex through one edge. Although edges can exist between two and only two distinct vertices, there is no limitation for the number of edges connected to each vertex. This reflects the physical wires and switches perfectly, as each switch has to exist between two and only two wires whereas wires don't have a limitation on how many wires they can connect to via switches.

In this model, CABs are no longer part of the RRG and not involved in placement and routing; instead they serve as organizational groups for components. Components are directly involved in setting the device and net constraints. Figure 2.13 shows how each component pin is paired with a RRG vertex that corresponds to a physical wire connected to the same component pin in actual FPAA. So, all interaction between placement and routing is captured via the component pin-wire pairings created from the architecture description. This is a simple yet effective model that allows many algorithms to be employed easily for placement and routing. As an added benefit, it can be used to specify arbitrary connections, which allows capturing and constructing a wide range of reconfigurable architectures sharing the underlying principle that can be summarized as: "every switch can connect two and only two wires." All topologies that follow this principle can be represented using this RRG model; therefore, a wide range of continuous-time FPAAs can be supported without

having to change the tool. We can specify any FPAA architecture generically using a device configuration file, the feature which inspired the G(eneric) in the name of GRASPER stands for.

A similar model has been used in wireC [69], Triptych [55], and Versatile Place and Route (VPR) [52] FPGA placement and routing tools. VPR is a popular FPGA placement and routing tool that offers flexibility in supported architectures thanks to the RRG model described above that can capture arbitrary switch-wire connections. VPR uses a technology-mapped netlist and a text-based FPGA architecture description file as input to generate a placement result for the circuit using simulated annealing [68]. In addition, VPR can continue to perform global routing or a combined global/detailed routing based on the Pathfinder negotiated congestion algorithm [70] for this placement result, or a pre-existing placement file that is read in.

Although limited to multi-driver routing approach, and a homogeneous CAB structure with wire segments that span only one CAB until version 5.0 [71], its flexibility made VPR the leading academic and commercial architecture exploration tool. VPR's success and widespread use by the FPGA research and design community has even secured a place to this tool in the SPEC2000 benchmark suite. However, it is designed primarily taking FPGA architectures into account, so does not really use the potentials of the RRG to its full extent. Degrees of freedoms granted when designing the interface took only the cases one would encounter in FPGA design. Although heterogeneous component support, which consists of additional digital components such as memory and multiplier, was added recently [71], VPR doesn't have transistor level simulation support yet; only digital delay calculation models, and cannot support the CAB types we need. VPR allows only a regular rectangular array of one block per each grid on a regular 2-dimensional array.

VPR input interface allows blif and verilog formats. These formats, as already discussed in Section 2.2, are not suitable for effectively describing analog circuit netlists, which is the main objective in this project. The range of interconnect topologies and CAB

array geometries supported by the GRASPER interface are wider. One difference between the FPGA-based earlier tools and GRASPER is that the FPGA-based tools capture the wire-switch relationship as a directed graph whereas GRASPER uses an undirected graph, since analog switches of interest are expected to be bidirectional. On the other hand, if unidirectional switch support is of concern, it is possible to extend the undirected graph-based data structure to a directed graph by replacing the undirected edges with two directed edges of opposite polarity. By doing this, all paths that could be found in the undirected graph will be available in the directed graph as well. Figure 2.14 illustrates this concept. This may be a useful addition when a mixed-signal analog-digital architecture support is considered in the future. Although it was not the main objective initially, it is possible to extend GRASPER to place and route digital circuits as well, but VPR does not have support for analog circuits.

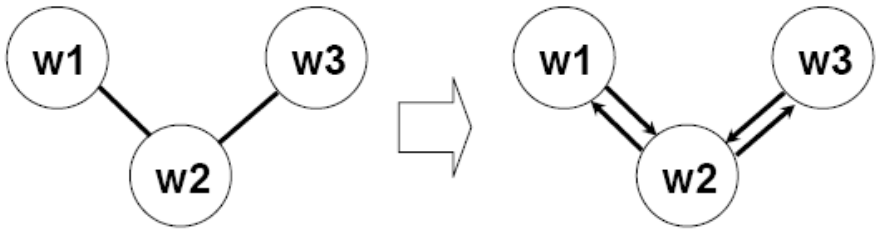


Figure 2.14. Converting an undirected graph to a directed graph. Each edge in the undirected graph is replaced with two directed edges, so that it is possible to find in the directed graph all the paths that exist in the undirected graph.

2.5.1 Placement in GRASPER

Because of the increased wire type variety, placement results no longer enforce a unique routing solution in GRASPER. Therefore, placement and routing are divided into separate steps again.

In a similar fashion to RASPER, GRASPER placement also uses MHEC-based cell ordering to maximize the use of local wires for routing. The details for this ordering is already mentioned in Section 2.4.1. Unlike RASPER, GRASPER lets only the components that match the cell type to be visited instead of the CABs. Each component that is feasible

for placement is ranked for placing the current cell, and the highest rank component is selected for placement after all components are visited.

Each pin of a CAB component is associated with wires that are connected to the routing resource graph as depicted in 2.13. Therefore, when a cell is placed into a CAB component, net information is instantly transferred to these pin wires. As pins of a net are placed on various locations on the chip surface, a bounding box, which is called as netbox, begins to form defined by the positions of the wires associated by this net. The larger this netbox becomes, the more distance is likely to be traversed by the wires in the routing phase. In addition, since larger netboxes may occupy more routing resources, they may also increase the chances of congestion. This concern brings another placement objective, which is the minimization of the sum of netbox sizes.

The quality of the placement results can be influenced in a certain direction to achieve different goals. Depending on the placement priorities, candidate components can be awarded or penalized in different ways during the ranking process. In the current implementation of GRASPER, increased CAB utilization is aimed and awarded. For a different case, this may not be an important issue and the award can be reduced relative to a different placement objective.

2.5.2 Routing in GRASPER

During the GRASPER placement step, each placed component pin that is connected to a wire is associated with a net defined in the circuit graph. These wires act as the terminals distributed to various edges of the RRG. The objective in routing is to find disjoint minimum spanning trees (MST) within the RRG for all nets where each terminal of a net is also a member of that net's MST. The concept of routing MST on an undirected graph is illustrated in Figure 2.15.

As mentioned in Section 2.5.1, net congestion can cause problems when routing each net. To reduce the chances of net congestion, netbox sizes are minimized during the placement phase. The order nets are chosen for routing also affects the success of routing. If

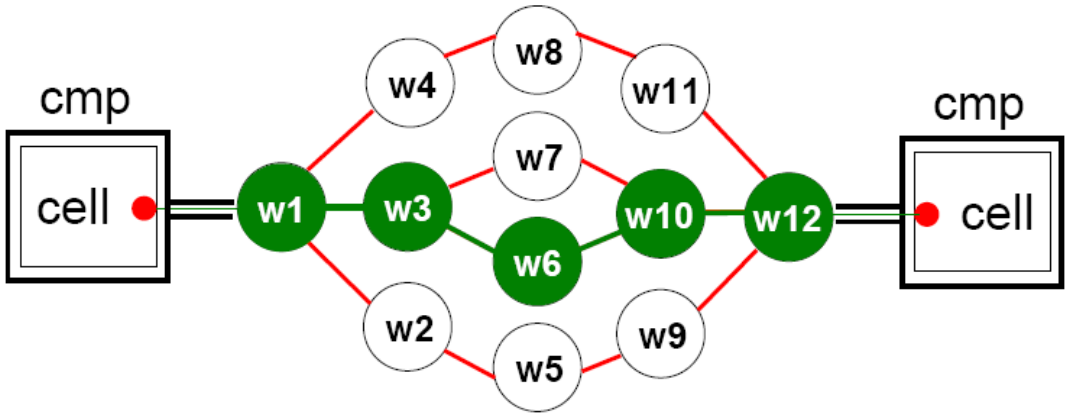


Figure 2.15. Grasper routing on the undirected routing resource graph. Source and sink terminals for each net are determined at the end of placement, routing establishes a minimum spanning tree that connects all terminals together.

nets that have larger netboxes are routed first, chances of congestion will increase for the subsequent nets. To avoid this, nets are listed in ascending order of size of their netboxes and routed in this order.

GRASPER routing uses the maze router approach based on Dijkstra's shortest path algorithm [72]. Maze router has two important strengths: first, it guarantees to find a connection between two terminals if it exists; and second, it guarantees minimum path. Its weakness lies in its tendency to radiate toward all directions from a source terminal, which causes large memory demands and time consuming operation. However, these weaknesses will not be as effective for the reconfigurable arrays as in the case of full custom design, because of the limitation on the number of the routing resources and their connections. In addition, the wires corresponding to the vertices in the RRG of GRASPER correspond to longer distances than the equivalent grids in the full custom design. This reduces the number of layers or wavefronts that are created at every propagation, resulting in finding the target terminals faster. This simple method works effectively in GRASPER.

The first step for routing each net is to determine the source and sink (target) terminals. Since there is no strongly defined signal flow direction in analog circuits, any wire can be selected as the source terminal. To minimize the number of propagations in RRG, GRASPER chooses the source terminal as the wire that has minimum distance to the center

of the netbox.

The next step in routing is wire labeling. In the labeling step, layers of wires are formed starting with the source layer that consists of the source terminal alone. In the beginning, source layer is the top layer since it is the only layer. Then, all wires adjacent to the wires in the top layer are labeled with a value that reflects the total distance from the source sink and added into the next layer. This can be a value derived from the geometric distance or wire capacitances involved depending on how routing cost is modeled. GRASPER currently uses the number of switches on each wire as the cost of wire. This count gives an approximate equivalent to the wire capacitances as well. Of course, only wires that are not already labeled or are not used for routing other nets can be labeled. When there are no more wires available for labeling among the adjacent wires of the top layer, next layer is moved on top of the layers becoming the new top layer. Propagation of wires and adding new layers continue until all sink terminals are labeled.

The final step in routing is backtracing from the sinks to the source. Backtracing is applied to one sink terminal at a time. Among the labeled adjacent wires of the sink wire, the wire with the least label value is chosen and added to the path. This continues until one of the adjacent wires is a wire that has already been added to the routing path for the current net. Obviously, source terminal is added to the routing path from the beginning, so that backtracing the first sink terminates at the source terminal wire. The same process is repeated for the remaining sink terminals.

2.5.3 GRASPER Placement and Routing Results

GRASPER returns an output file that is essentially a list of switch coordinates used by the programming interface to program routing switches (RSW), bias or configuration switches (CSW), and switch elements (SWE). To observe the GRASPER routing results visually, a program called Field Programmable Analog Array Routing and Analysis Tool (FPAA RAT) can be used. FPAA RAT was developed by David Abramson and Scott Koziol in the CADSP Research Group of Georgia Tech. FPAA RAT can be used for either viewing the

wires used for routing the placed components on FPAA, or for changing the switch list by manually adding or removing switches on the user interface. FPAA RAT currently supports RASP2.8 architecture, but is being extended to support future ICs in the RASP family. Figure 2.16 demonstrates the GRASPER placement and routing results obtained for an 8th order elliptic band-pass gmC filter (ebpf8) that contains 25 OTAs and 16 capacitors. Since RASP2.8 contains at most one OTA in each CAB, 25 CABs had to be used to complete the placement. Placement and routing statistics for this circuit as well as three other band-pass gmC filters are presented in Table 2.2.

Table 2.2. Placement and routing statistics for 8th order band-pass gmC filters mapped to RASP2.8 using GRASPER.

circuit	components	nets	CABs	wires	RSW	CSW
bbpf8	33	12	17	141	128	17
c1bpf8	35	13	19	145	131	19
c2bpf8	41	13	25	179	165	25
ebpf8	41	13	25	179	165	25

2.5.4 SWE Routing in GRASPER

SWEs are also circuit elements of special interest in the routing stage. By definition, each SWE connects two distinct nets with a routing switch programmed to the characteristics desired by the user. Since SWEs do not correspond to actual CAB components but are only part of the routing fabric, they can not be dealt with during the placement stage; and can only be realized during the routing stage. Routing a SWE is in fact establishing a routing path between two disjoint MST for routing two different nets. If both nets connected to a SWE are already associated with at least one wire, a routing path can be established between wires of each SWE terminal in a similar fashion to routing the other nets, marking one of the switches on the path as the intended SWE. This algorithm works only when all net terminals are connected to a wire; therefore, SWE routing is performed as the last step to wait until all nets are routed and their wires are established.

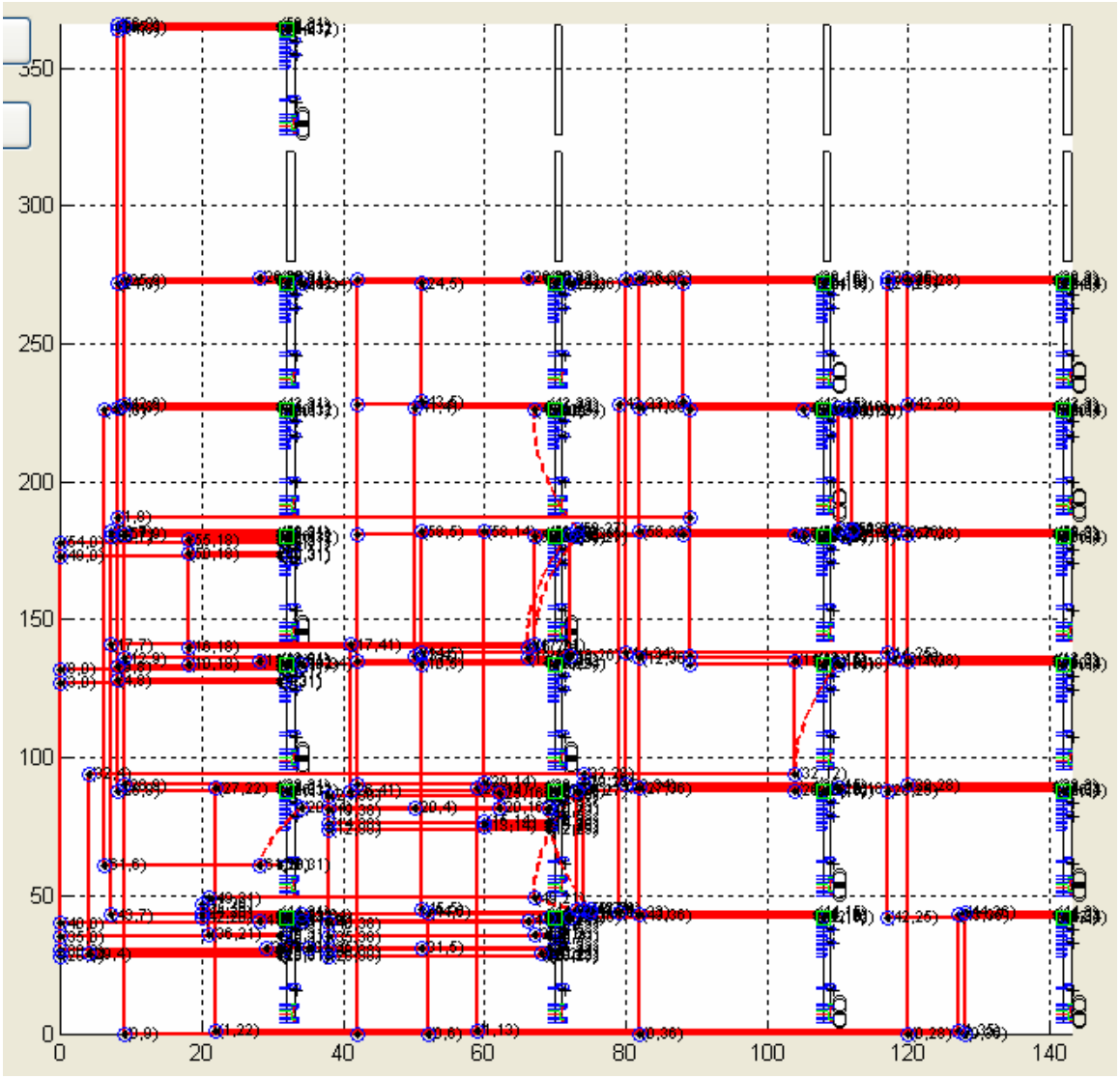


Figure 2.16. GRASPER routing results for an 8th order elliptic band-pass gmC filter as viewed by FPAA RAT.

The algorithm described above cannot complete SWE routing for all circuits. Take the diffusor circuit illustrated in Figure 2.17 as an example. In this circuit, three of the nets (4, 6, and 8) are not connected to any CAB components or an I/O pin; therefore, they are floating nets and cannot be routed using the method described above. A floating net is a net in the circuit that is not associated with any wires before the SWE routing stage. In this case, all SWEs that do not have a floating net are routed first, followed by the SWEs that have one floating net only. To route a SWE with one floating net, it is sufficient to select a switch that is connected to any wire that is associated with the non-floating net of that SWE, and has a wire that was not already used for routing another net on the other side. This wire on the other side of the selected switch is the previously floating net of the SWE, which is no longer a floating net since it is associated with a wire now. SWEs with two floating nets cannot be routed before associating at least one of their nets with a wire.

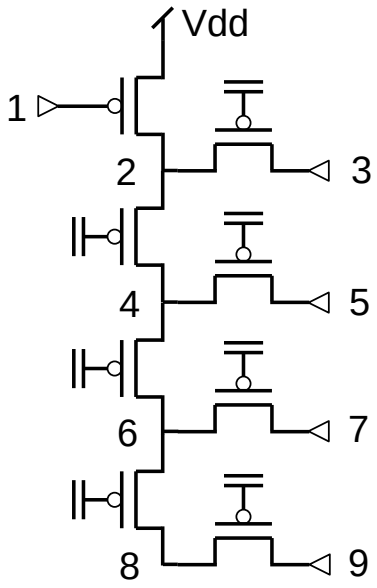


Figure 2.17. A diffusor circuit with one PFET and 7 SWEs. Nets 4, 6, and 8 are floating nets, since they are not connected to I/O pins or any CAB components.

To complete SWE routing using the methods described above, it is necessary to order the SWEs in ascending order of the number of floating nets they have. SWEs with no floating nets must be routed first, followed by the SWEs with only one floating net. As

SWEs are being routed, some floating nets will become non-floating nets, so SWE routing order must be updated for those SWE that are connecting to these affected nets. Eventually, there will be no SWEs that have two floating nets, and it will be possible to route all SWE this way.

Since using the routing switches as components has been possible only with the floating-gate FPAAs, the problem of SWE routing has not been addressed and solved previously. Therefore, SWE routing algorithm is a novel contribution of this work, which leads to a significant increase in the utilization of the switch fabric.

2.5.5 GRASPER Placement and Routing Results for Vector Matrix Multiplier (VMM) Circuits

In this section, the placement and routing efforts for the vector matrix multiplier (VMM) circuits are summarized. VMM circuits demonstrate how the switch fabric can be utilized as active computation elements, which is a unique feature of the floating-gate based FPAA. These circuits can be built using only 1 OTA for each row of the coefficent matrix, only requiring switches to store coefficients in different columns. Figure 2.18 depicts a 2*2 VMM circuit that has 2 OTAs and 2 SWEs to the left of the OTAs for receiving the input signals in current mode, and 4 SWEs on the right side that store the coefficients for the matrix multiplication. The number of SWEs on the right side of the OTAs must be equal to the multiplication of number inputs and outputs, whereas the number of OTAs, and the number of SWEs to the left of the OTAs are equal to the number of inputs. The operation of the VMM circuits built using SWEs are detailed in previous studies [73].

One can see the rapid increase in switch requirements as the size of VMM grows. Incorporating the SWE routing algorithms described in Section 2.5.4, GRASPER was initially able to route a maximum size of 10*10 VMM circuits as illustrated in Figure 2.19. One observation made on these results was the presence of routing congestion in a certain portion of the IC while other regions remained unused. This was a direct result of the placement of OTAs, which was highly affected by the I/O pin choices, which had the tendency to cluster

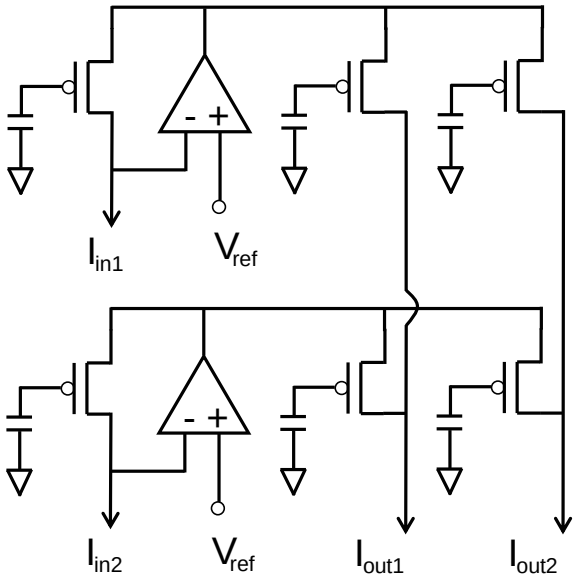


Figure 2.18. A 2*2 VMM circuit that uses 2 OTAs and 6 SWEs.

to a certain corner of the IC. To balance the OTA placement, the I/O pins in the circuit netlist were reordered, so that they were spread more evenly across the IC. The immediate effect of reordering was the ability to route 14*14 VMM circuits, which is illustrated in Figure 2.20. This routing result left less unused areas on the IC, although it was observed that the OTAs still had the tendency to be placed toward the bottom left corner of the IC, increasing the congestion in this area. This effect was caused by the CAB ranking when placing the OTAs, which favored the order followed when ranking the CABs. Finally, alternate CAB rank orderings were introduced; where it is possible to rank CABs from bottom left to top right or vice versa, resulting in a tendency to place CABs to either corner. A third ordering scheme alternates between these two orderings to further balance the OTA placement, and achieves successful routing of a maximum size of 15*15 VMM circuits, as illustrated in Figure 2.21. Placement and routing statistics for each of these VMM circuits mapped to RASP2.8 using GRASPER are presented in Table 2.3.

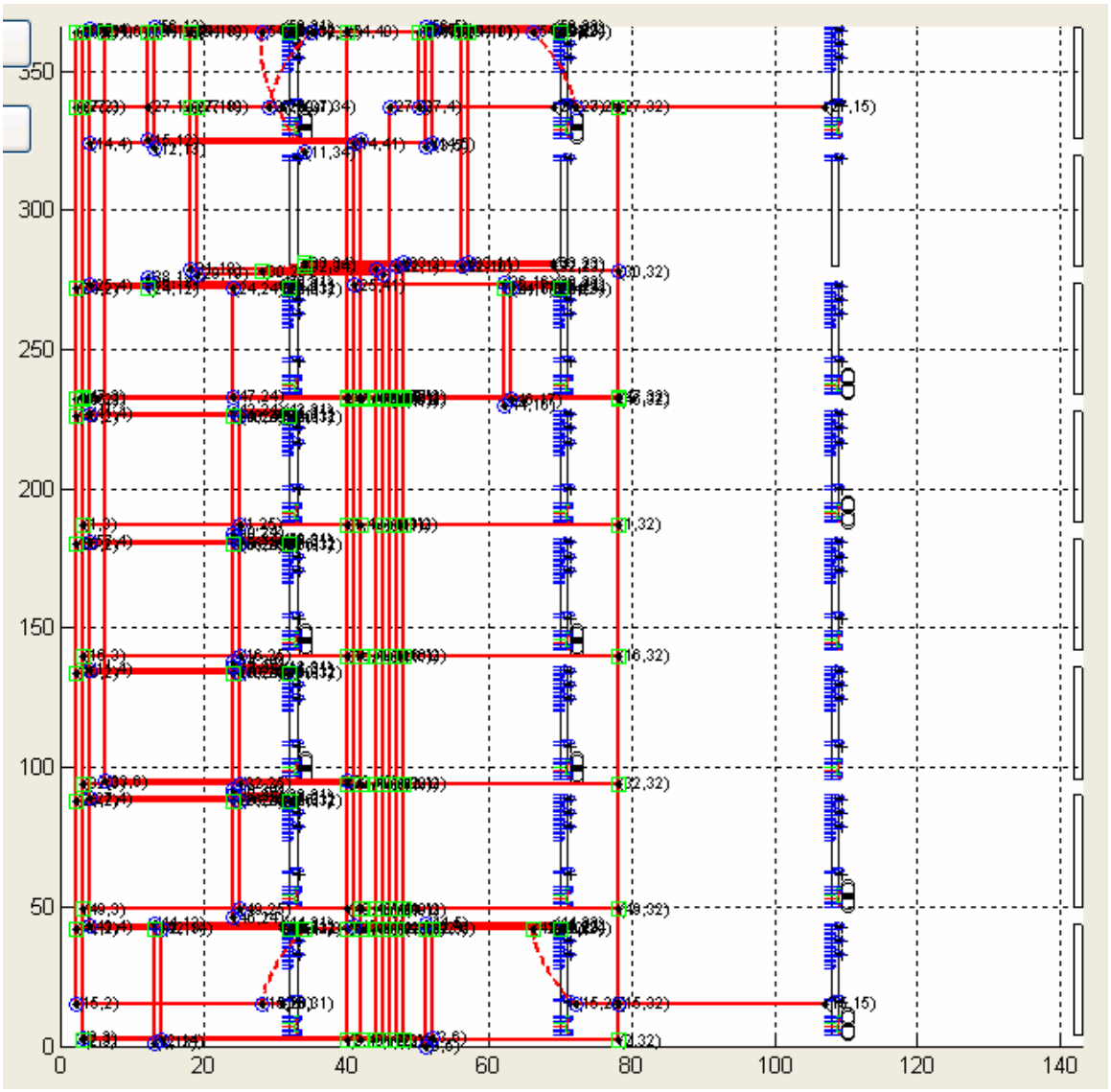


Figure 2.19. GRASPER routing results for a 10*10 VMM circuit as viewed by FPAA RAT.

Table 2.3. Placement and routing statistics for different sized VMM circuits mapped to RASP2.8 using GRASPER.

circuit	components	nets	CABs	wires	RSW	CSW	SWE
vmm10*10	10	33	10	107	74	10	110
vmm14*14	14	45	14	215	171	14	210
vmm15*15	15	48	15	248	200	15	240

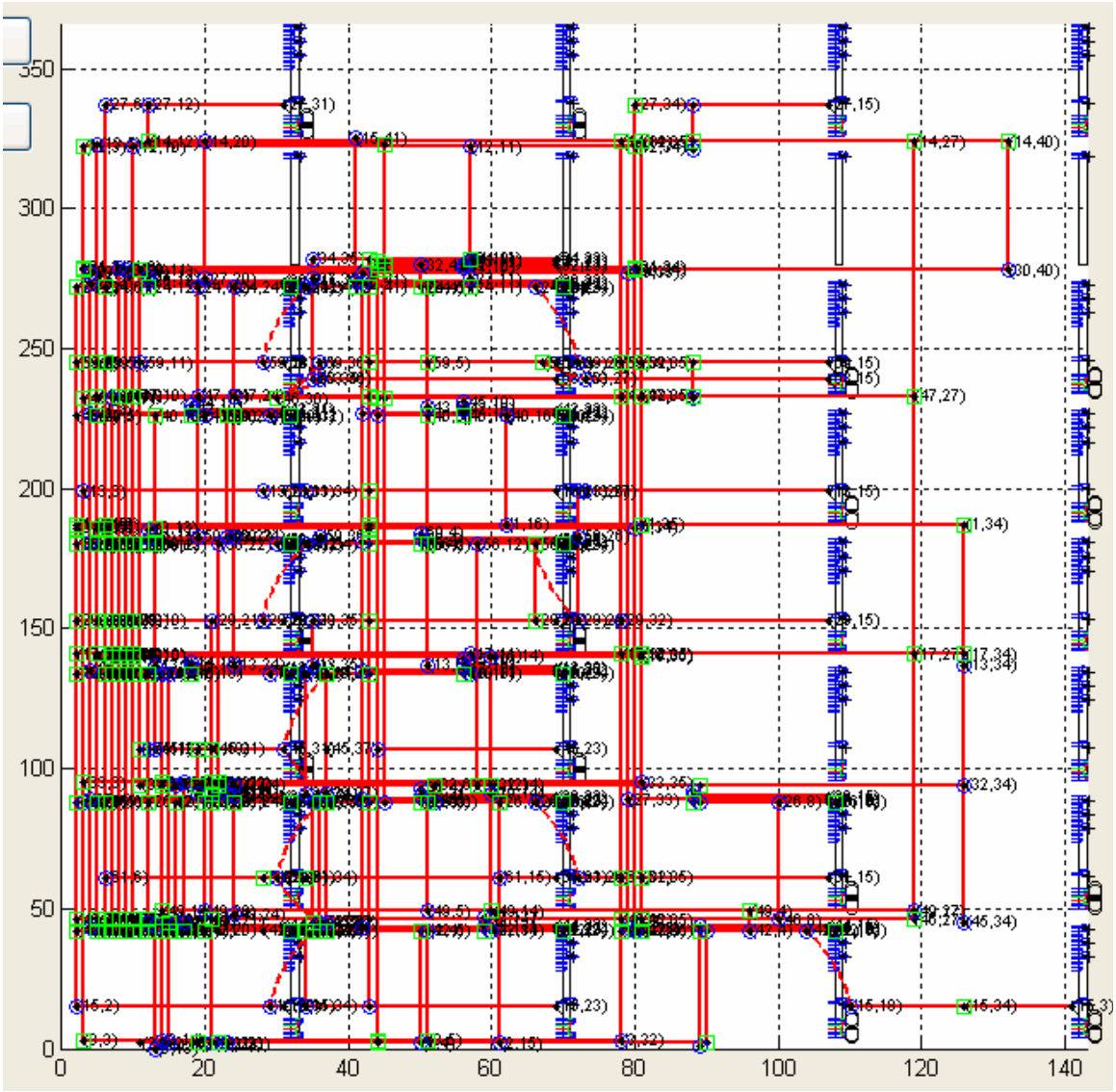


Figure 2.20. GRASPER routing results for a 14*14 VMM circuit as viewed by FPAA RAT.

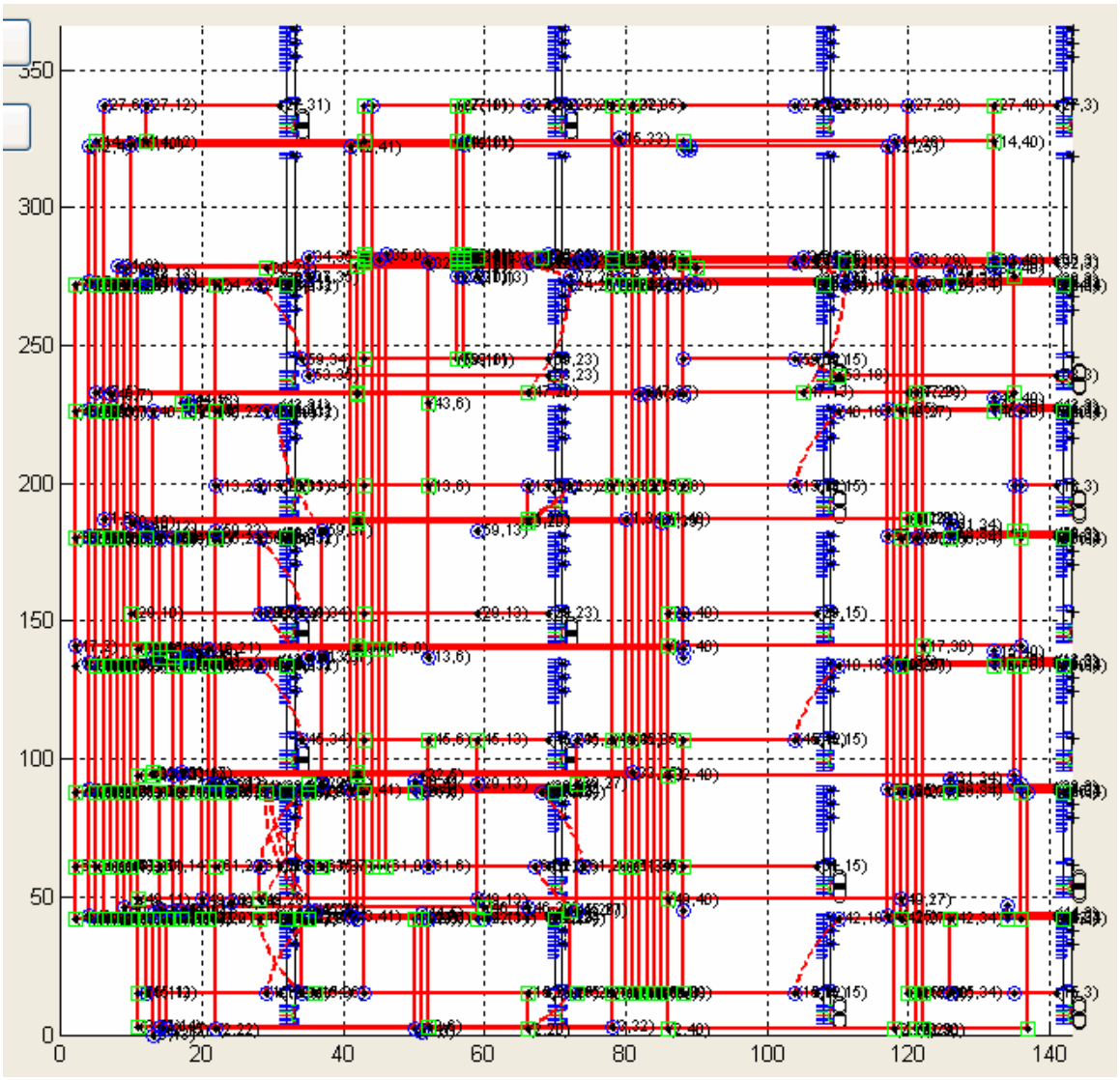


Figure 2.21. GRASPER routing results for a 15*15 VMM circuit as viewed by FPAA RAT.

2.6 Impact of the Placement and Routing Tool on FPAA Research

Having an automated placement and routing tool makes the FPAA technology more accessible. Users of FPAA should be focused on developing analog systems for performing high level tasks. GRASPER provides an abstraction of the routing infrastructure from the users, so that they don't have to deal with the low level tasks such as optimum placement of components on the CABs or switch list generation. Design and implementation of more complex systems is only possible when the users can be relieved of such low level tasks.

With the confidence developed by building and testing larger systems on the FPAA, the architecture research can also be pushed forward. The generic architecture support of GRASPER has also been useful in developing new architectures. Currently, a set of architectures commonly named as RASP2.9 is under development, and they are also supported by GRASPER by means of a device configuration file. RASP2.9 in general uses similar CAB structure to RASP2.8, but with an array of 6 rows and 14 columns, and some variations of RASP2.9 uses reduced number of switches on the crossbar. Architecture configuration files for some architectures in the RASP2.9 series have been written and successfully tested. GRASPER can also take the resources and the process technology of a new FPAA into account, and incorporates them into the placement and routing, followed by the parasitic extraction; therefore, allows evaluation of new architectures by simulation of circuits on these architectures even before the IC is fabricated. This will make planning new architectures easier and reduce the costs of designing new FPAAs.

At present, GRASPER is being used by the students of the CADSP Research Group and also for educational purposes by the students in various courses in Georgia Tech, which makes about 35-40 student users at the time. In addition, it has also been used in workshops by attendees from different universities and research institutions. The tool has served well in these platforms, and the feedback received from the users has been used to improve its usefulness and robustness. When the FPAA hardware platforms become commercialized, GRASPER will be used by a wider group of people and institutions for both applications

and research.

CHAPTER 3

SIGNAL INTEGRITY AND PERFORMANCE OPTIMIZATION

Parasitic components coming from routing using non-ideal interconnects have an adverse effect on the performance of a circuit implemented on any reconfigurable device. In digital design this effect may be observed as reduced performance, but in analog design even the functionality may be compromised. Interconnects on the circuit path inevitably deteriorate the signal integrity; therefore, the effects of interconnects on the circuit performance must be studied carefully and taken into account to minimize this deterioration. The need for an automated PNR tool becomes more obvious when these tasks that are very difficult to manually perform are considered. This justifies the use of such a CAD tool for the relatively smaller analog circuits.

Although circuits can be directly evaluated on the FPAA after PNR, being able to know immediately by simulation without having to map every attempt on the FPAA is more convenient, faster, and more suitable for automation. This is especially useful when it is desired to optimize the circuit performance with little or no human interaction.

Another use of evaluating circuits is for debugging purposes. One can not probe into the intermediate nets in a circuit when it is implemented on the FPAA. But it is possible to track all points in the design using a simulator, which may give us valuable hints in troubleshooting.

Finally, it is useful to have this evaluation ability for the architectures that are not readily available yet. Chapter 4 presents a systematical method to test how the performance of various circuits on conceptual architectures can be evaluated before even designing and fabricating those architectures.

3.1 Parasitic Extraction in RASPER

In RASPER, each interconnect is defined by a combination of switches and wires associated with nets and switch elements. The wires are called in order of the most local to most global as horizontal local (hl), vertical local (vl), vertical global (vg) and horizontal global (hg) wires. The scopes of different wire types have been described in Section 2.3.1. Each wire is connected to the other wires through a switch while the wires themselves form a tree with the most global wire being the root and the most local wires being the leaves. Each wire type is saved in an interconnect library as a subcircuit along with the switch. The advantage of using this flexible library structure is that it can easily be replaced by another library containing different switch and wire models if desired and the change in the behavior can be observed. In the simple interconnect model used in this work, each switch is approximated by a resistor and each wire by a capacitor. Users are welcome to improve these models to any level of accuracy they desire keeping in mind the cost of increased simulation times.

To reflect the impact of wires and switches on the circuit performance, a subcircuit is generated for each net as depicted in Figure 3.1. This subcircuit as a block replaces the net in the actual circuit, and all component pins that used to be connected to the original net are connected to this block instead. Simulating this circuit in SPICE, synthesis results can be verified with parasitic effects included. Incorporation of the interconnect model in simulation helps in troubleshooting, as well. It gives access to every node in a circuit to determine the causes of a problem after the circuit is synthesized. This wouldn't be possible on the programmed chip because of limited availability and the parasitic effects of I/O pins.

3.1.1 Measurements vs Simulations Using a Band-pass Filter

Seeing a comparison of circuit measurements against simulations is helpful to verify the interconnect model developed for RASPER. Filters are a class of commonly designed analog circuits, so the band-pass filter (BPF) presented in Figure 3.2 is implemented to do this measurement-simulation comparison. This filter uses a simple topology that consists

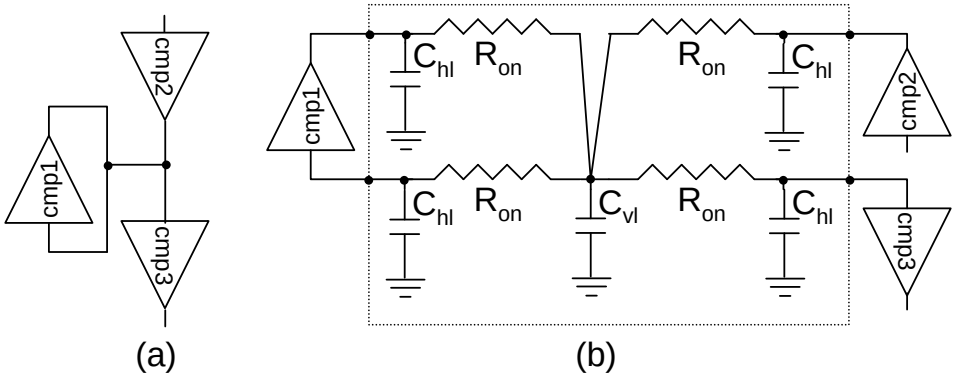


Figure 3.1. Subcircuit block that replaces the interconnect in the circuit.

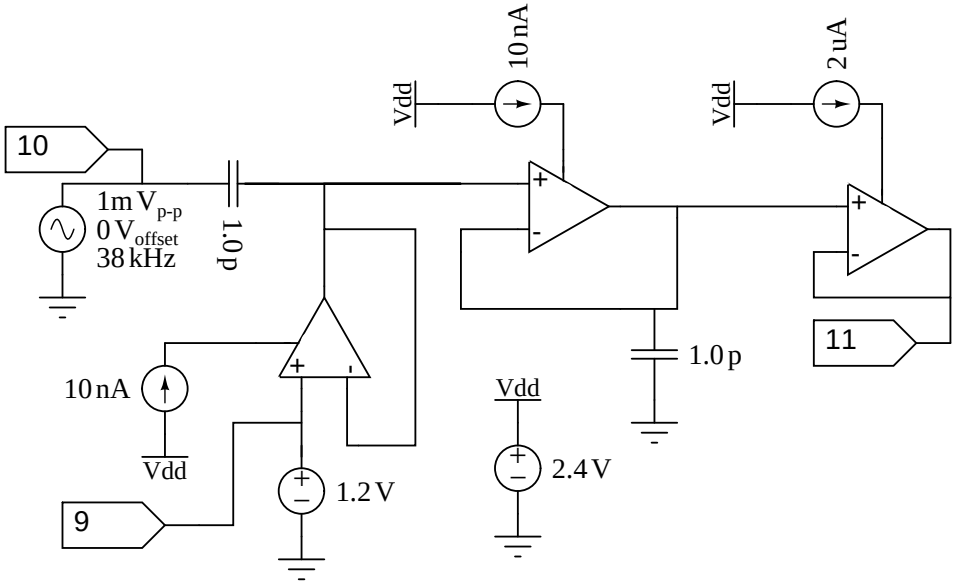


Figure 3.2. Band pass filter made from a cascade of a high-pass and a low-pass filter.

of only two passive capacitors and three OTAs each of which being used as a high-pass, low-pass and buffer stages. Each stage allows control of the corner frequency through the conductance (Gm) of their respective OTAs using the bias currents, so the design parameters can be easily adjusted. It is also a function that is highly reused in more complex designs; therefore, a good candidate to conduct some experiments on.

One should keep in mind that the simulations may not completely predict the actual implementations of analog circuits in hardware. Various factors influence this deviation, such as process variations, mismatch, etc. Even the switches at a certain region of the chip

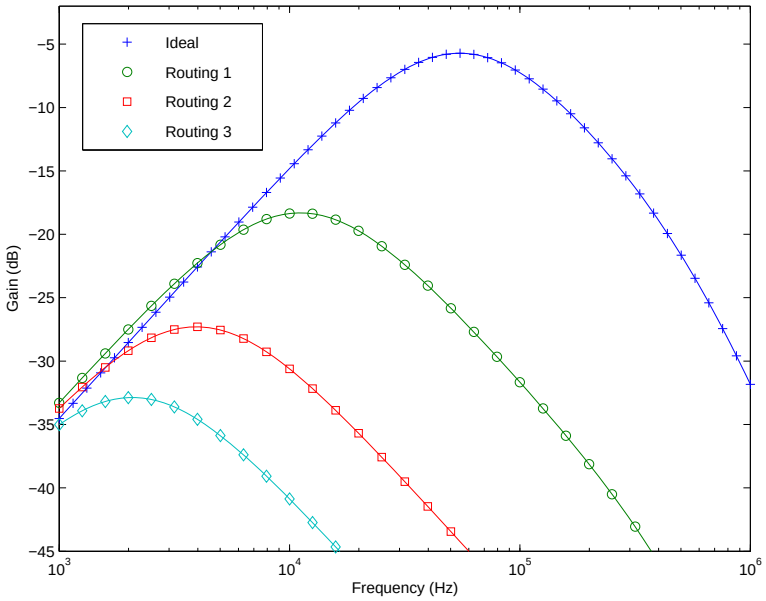


Figure 3.3. Simulations of the BPF circuit for ideal interconnects (ideal), all OTA in the same CAB (routing1), all OTA in the same column but in different CABs (routing2), and all OTA in different columns (routing3).

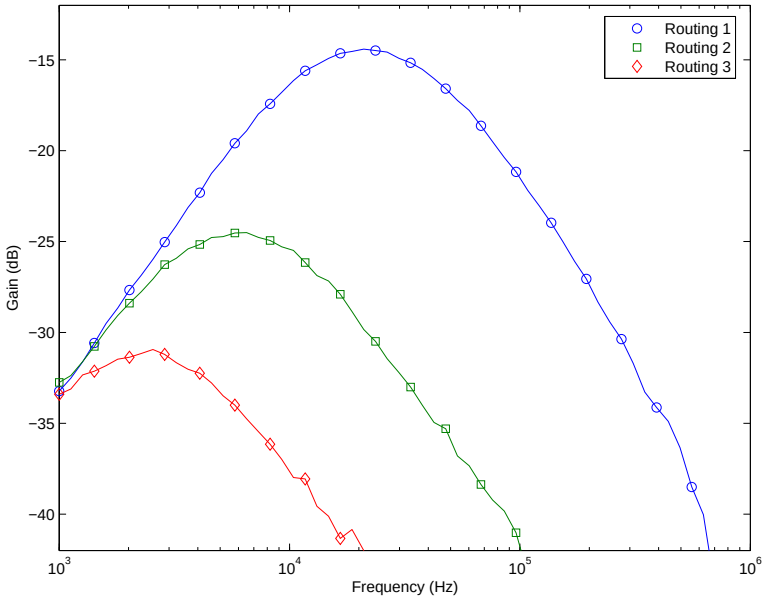


Figure 3.4. Measurements of the BPF circuit for all OTA in the same CAB (routing1), all OTA in the same column but in different CABs (routing2), and all OTA in different columns (routing3).

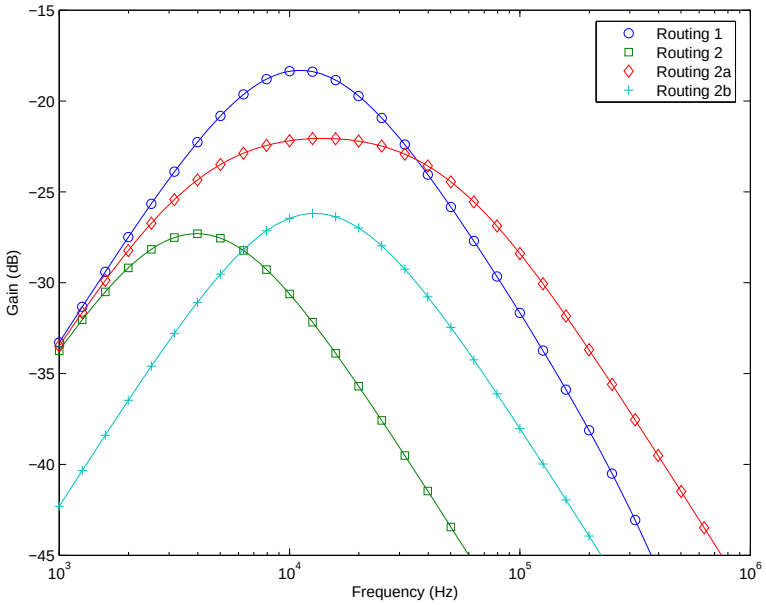


Figure 3.5. Simulations of the BPF circuit for all OTA in the same CAB (routing1), and all OTA in the same column but in different CABs (routing2). The second configuration is modified to approximate the first configuration by adjusting the low-pass corner only (routing2a), or both corners to achieve the same Q factor at a lower gain (routing2b).

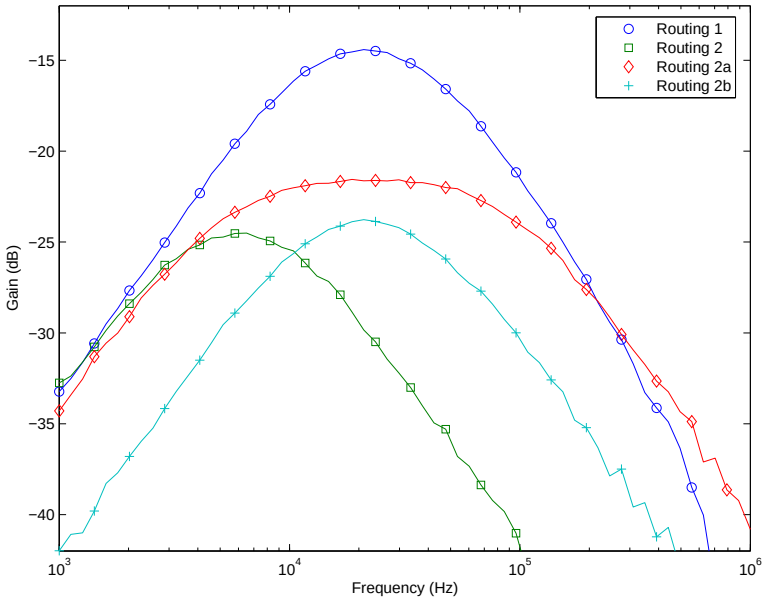


Figure 3.6. Measurements of the BPF circuit for all OTA in the same CAB (routing1), and all OTA in the same column but in different CABs (routing2). The second configuration is modified to approximate the first configuration by adjusting the low-pass corner only (routing2a), or both corners to achieve the same Q factor at a lower gain (routing2b).

may behave slightly different from the switches in another region. However, one would care less about the accuracy so much as functionality, since the accuracy of the simulations depends heavily upon the transistor models matching the actual devices. The intention is to give the user the ability to predict the outcome of the circuit, a tool that can be used in troubleshooting and later as a guide to optimization. The tool can also be used to determine the effect of the routing parasitics upon the ideal circuits, which allows the user to adjust the circuit topology or biases to compensate.

The initial results displayed in Figures 3.3 and 3.4 are intended to show comparisons between three different routing solutions for the same circuit. Among these, routing1 is a solution where all components are packed closely in one CAB. In routing2 they are distributed into different CABs of the same column. Finally in routing3 they are placed into different columns which will give the longest distance between them. A simulation of the circuit with actual FPAA components and ideal interconnects is also given as the ideal case in Figure 3.3. Note that in the current FPAA technologies, total delay in a path is strongly dominated by the switch parasitics rather than wire length; therefore, once a connection is made to any other column, it suffers the most degradation [6]. As one can observe from Figures 3.3 and 3.4, simulation results follow the actual measurements closely with minor errors. Since the interconnect model is modular, it can easily be replaced by a more accurate model to further reduce the deviation depending on the required level of accuracy.

Circuit performance between various routing solutions can be significantly different for a given set of bias conditions, as seen in Figures 3.3 & 3.4. Some of this may be compensated by tuning the corner frequencies via bias currents of the OTAs. In Figure 3.5, the configuration in routing2 is modified in two ways in an attempt to reach the routing1 performance. The first solution simply adjusts the low-pass corner to better match the original shape of the routing1 configuration. The second solution adjusts both corners to better match the corner frequencies and Q of the routing1 frequency response, except with a lower gain. The same configurations are also simulated using the netlists generated by RASPER

and displayed in Figure 3.6. Table 3.1 summarizes some of the design metrics used to evaluate the various implementations and gives a comparison between the simulations and measurements described above. From these results, one can observe the reduction in the corner frequency and gain while Q is preserved as the routing configuration changes from routing1 to routing3. This is because of the fact that OTAs are placed most distant from each other in routing3 and closest in routing1, reflecting the higher impact of the parasitic interconnects for the more distant placement configurations. Obviously, corner frequencies and gains of all the routing configurations are much lower than the simulation results with ideal interconnects. The results for routing2a and routing2b demonstrate how routing2 configuration can be compensated to get corner frequency values closer to routing1 configuration by adjusting the OTA bias currents, at the expense of either Q or gain values.

Table 3.1. BPF metrics from simulation and measurement results, where center frequency is in kHz, and gain in dB.

	simulation					
	ideal	route1	route2	route3	route2a	route2b
fc	46.4	11.1	3.9	2.1	14.0	12.9
gain	-5.75	-18.32	-27.30	-32.88	-22.06	-26.19
Q	0.504	0.501	0.501	0.504	0.259	0.495
	measurement					
		route1	route2	route3	route2a	route2b
fc		21.4	6.2	2.0	22.2	21.4
gain		-14.41	-24.51	-30.94	-21.56	-23.77
Q		0.452	0.451	0.406	0.200	0.464

3.1.2 Parasitic Extractions of Sample Circuits in RASPER and Their Simulation Results

Parasitic extraction and simulation results have been obtained for several other circuits as well. Here, the effects of the interconnect parasitics on the circuit performance for an 8-bit digital-to-analog converter (DAC), a voltage controlled oscillator (VCO), and a winner-take-all circuit (WTA) were studied.

The DAC implemented here is a binary weighted type, where SWEs are put into use for programming the binary currents. Figure 3.7 depicts the circuit. Simulations were run on

the circuit before and after synthesis. Among various design metrics, finding the settling time that results in a conversion rate and integrated nonlinearity (INL) was of main interest. To find the INL, an ideal model of a DAC has been simulated with the actual circuits, and the output voltages have been compared. According to the simulation results in Figure 3.8, the metrics in Table 3.2 have been obtained. These results indicate that the precision of the DAC in this case is preserved when placed and routed on FPAA; however, it runs 5 times slower.

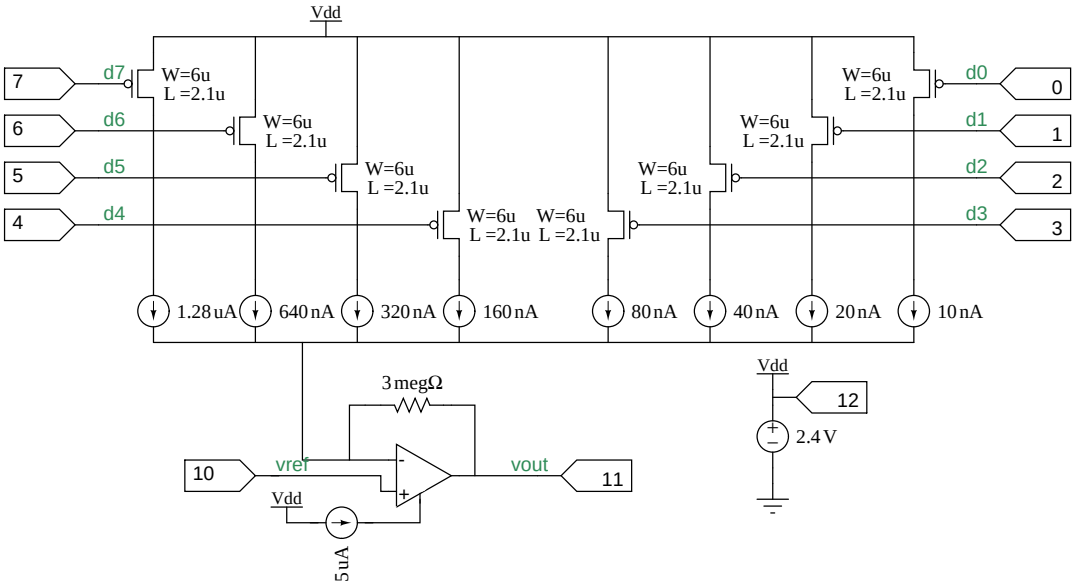
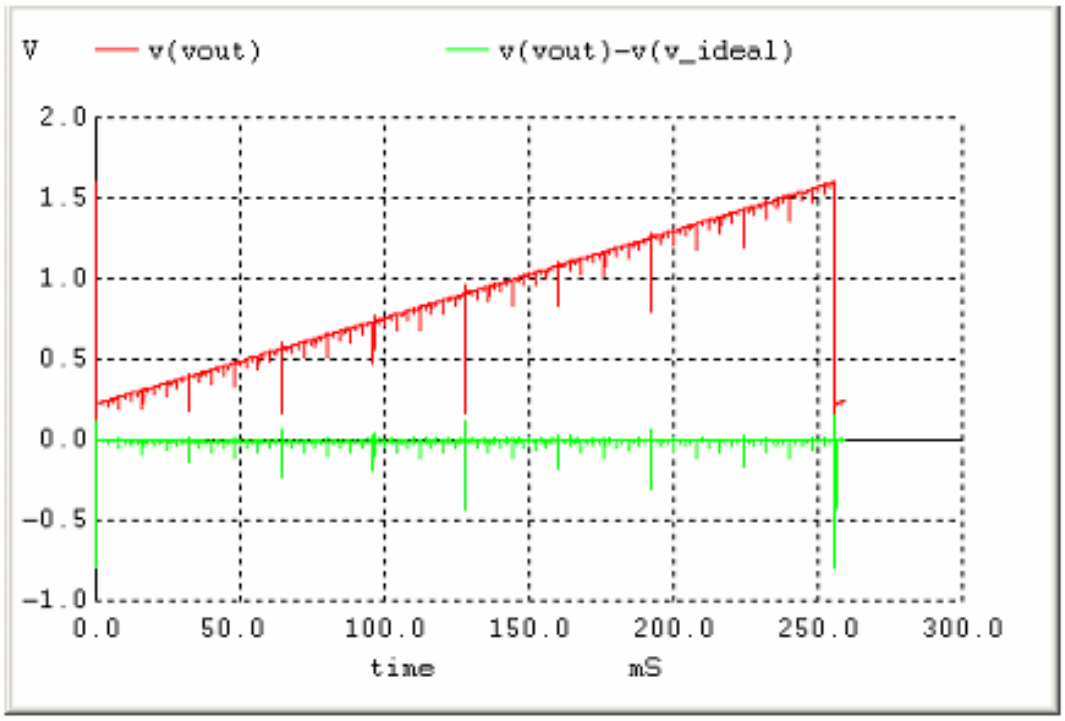


Figure 3.7. 8 bit DAC implemented with binary weighted SWE.

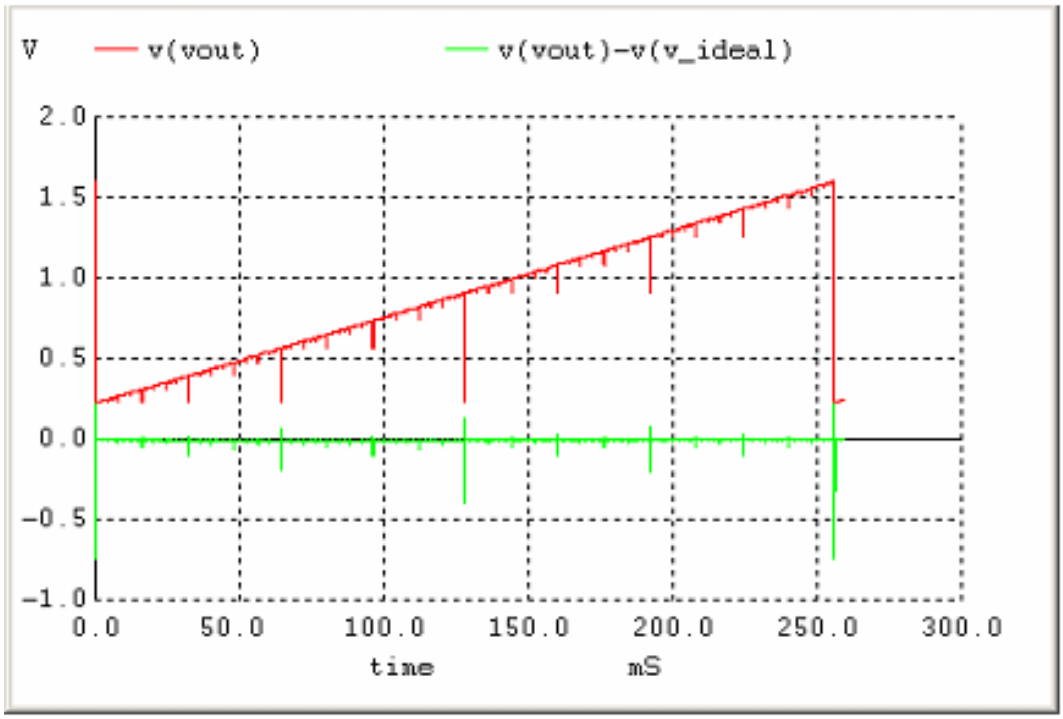
Table 3.2. DAC metrics

metric	pre-synthesis	post-synthesis
LSB	5.34 mV	5.34 mV
-INL (LSB)	-1.4	-1.5
+INL (LSB)	0.07	0.15
settling time (μs)	4.2	20

The VCO depicted in Figure 3.9 uses the ring oscillator structure, where each inverter is biased to a current controlled by the gate voltage of a current mirror. Increased current levels allow a higher frequency of oscillation and vice versa. The current flowing through the mirror responds to the gate voltage nonlinearly, further increasing the dynamic range of



(a)



(b)

Figure 3.8. Analog conversion of a sweep of digital values from 0 to 255 in an 8 bit DAC with (a) ideal interconnects and (b) synthesis parasitics included.

the oscillation frequency. Phase noise is an important metric in the characterization of an oscillator. However, since it is not possible to determine this metric through SPICE analysis, showing how the routing parasitics slow down the oscillation frequency was reported, as demonstrated in Figures 3.10 and 3.11.

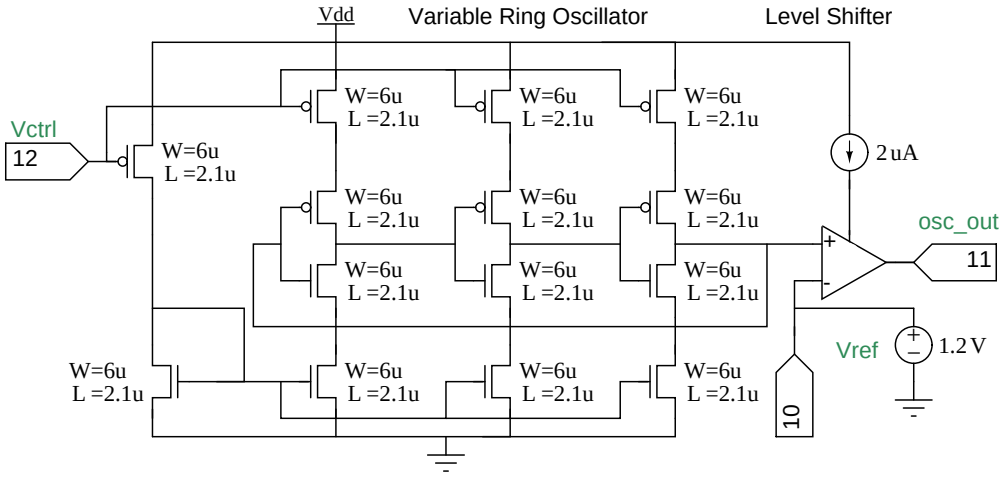


Figure 3.9. VCO with adjustable current inverters.

Winner-take-all circuits are common in analog signal processing because they can be used as a current comparator for an arbitrarily large number of input currents. Figure 3.12 shows the WTA circuit used for our analysis. The circuit determines which input current is the largest and outputs a voltage near the supply rail indicating the winning current input. An important metric of this circuit therefore is the smallest current difference that the WTA circuit responds to correctly. Simulation of the pre-synthesis circuit reveals that the discrimination ability is about 0.4 nA. Figure 3.13a shows the WTA circuit outputs as the result of a one-hot sequencing of the input currents for decreasing current differences of 0.4 nA, 0.2 nA, and 0.1 nA respectively. The desired result is for each output to take turns going high. The figure shows that for the smaller differences, the WTA circuit output does not have the desired result. A similar simulation was done for the post-synthesis case, but instead use larger current differences of 4 nA, 2 nA, and 1 nA in Figure 3.13b. In these simulations, the current discrimination ability was observed to be about 10 times

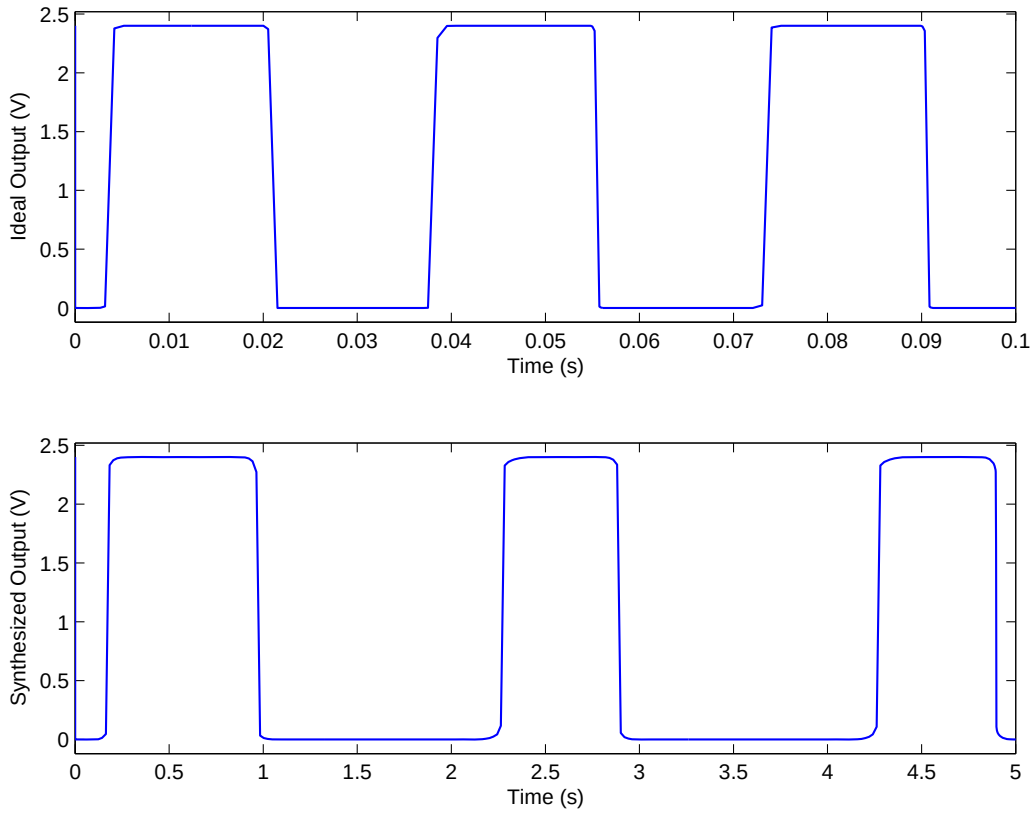


Figure 3.10. VCO output at Vctrl = 2.05 V where the oscillations just begin. Frequency of oscillation drops from 29 Hz to 0.5 Hz when parasitics are included.

less for the synthesized circuit. It should also be noted that the synthesized circuit output no longer reflects the symmetry seen in the pre-synthesized case as a result of the added routing components.

3.2 Performance Optimization Using Simulated Annealing

The interconnect models for RASP2.5 and RASP2.7 architectures defined in Section 2.3.1 show that the FPAA has three basic types of wires each carrying different parasitic values into the placed and routed circuit. It is possible to model routing interconnects by replacing each net between neighboring component pins with an equivalent subcircuit consisting of RC branches that connect its pins to each other to reflect the wire and on/off switch impedances as depicted in Figure 3.1. Simulation of the circuit with and without this subcircuit allows observing the impact of routing parasitics on the circuit performance. The

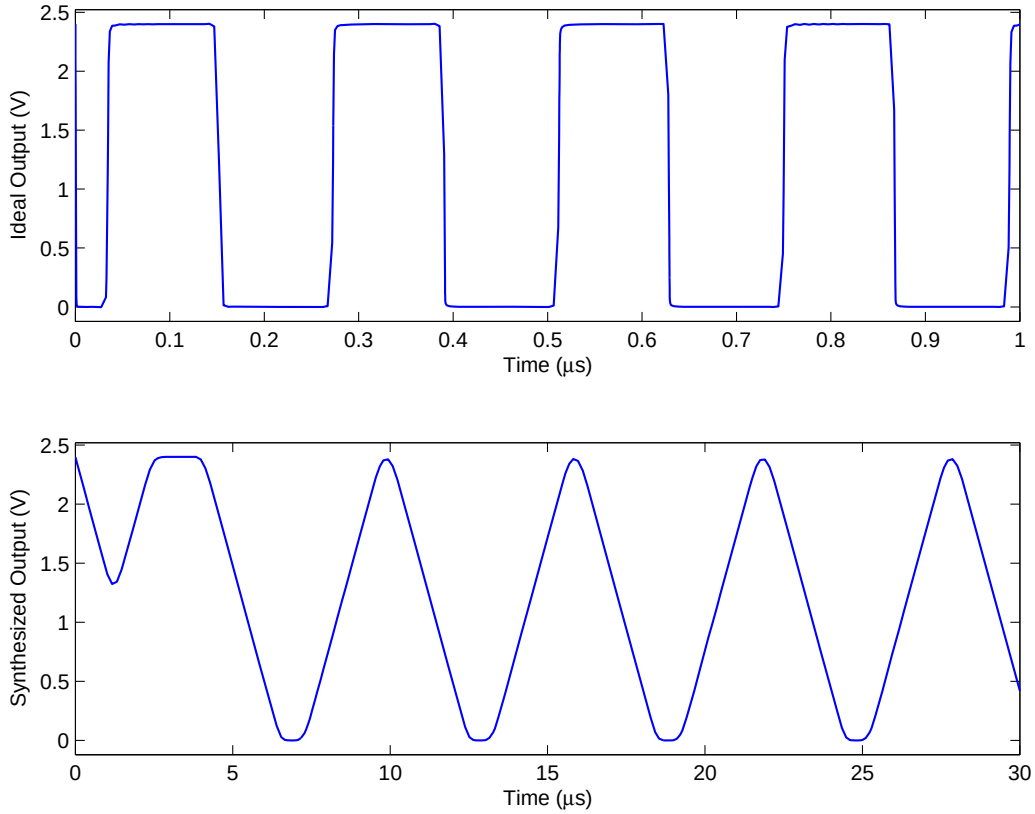


Figure 3.11. VCO output at Vctrl = 1.45 V where the maximum frequency of oscillation for synthesized circuit is attained at 167 kHz. Pre-synthesis circuit oscillates at 4 MHz with the same input.

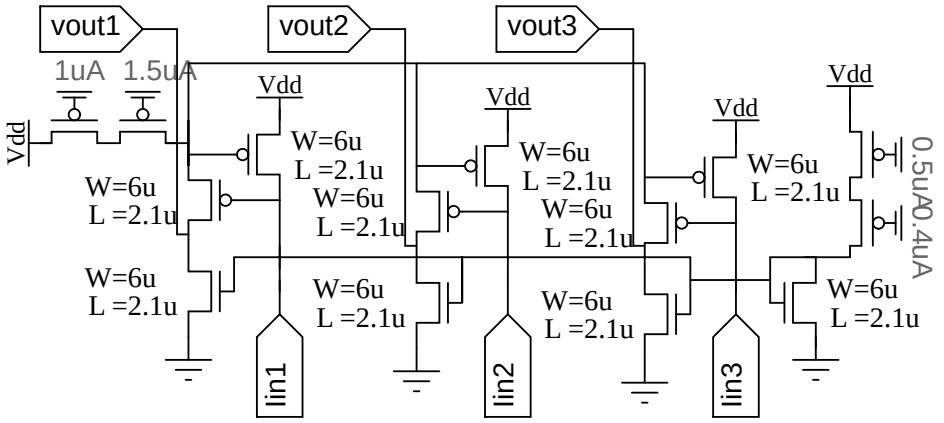
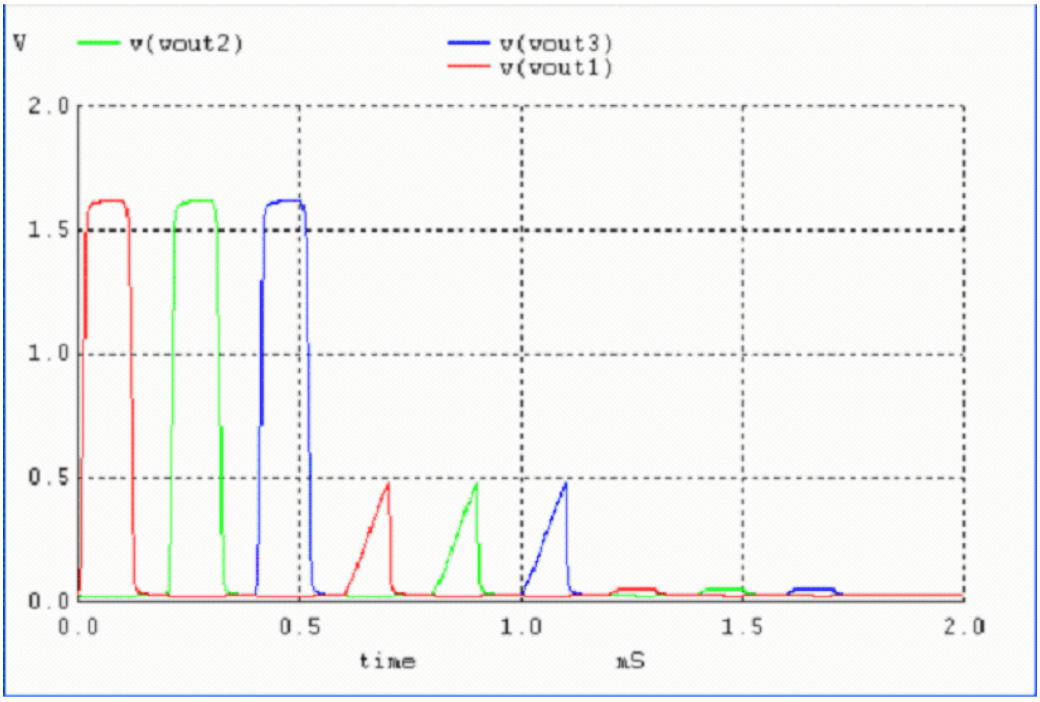
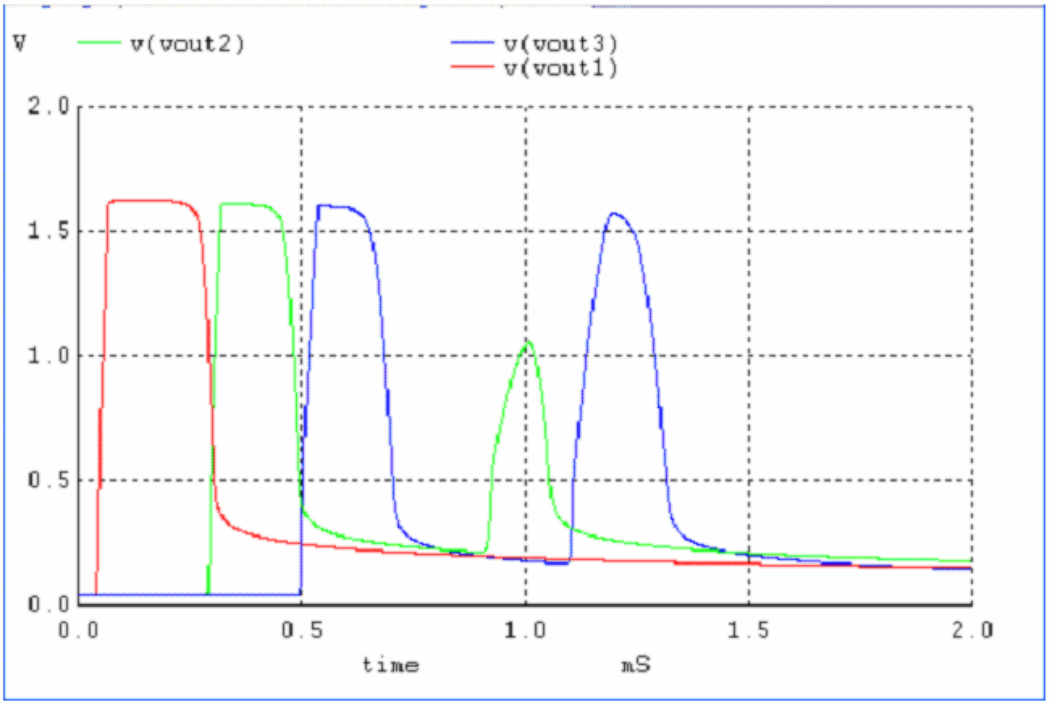


Figure 3.12. Schematic for a Winner-Take-All circuit with three current inputs.

accuracy of the simulation results depend on the sophistication of the employed models. In [74], a similar model has been used to compare simulation results against measurements



(a)



(b)

Figure 3.13. Winner-Take-All circuit simulation results for (a) pre-synthesis and (b) post-synthesis circuit netlists where one of the three input currents is larger than the other two inputs. Input current differences are 0.4 nA, 0.2 nA, and 0.1 nA for the pre-synthesis circuit where the outputs can be distinguished reliably for only 0.4 nA difference; while input current differences are 4 nA, 2 nA, and 1 nA for the post-synthesis circuit where the outputs are correct for only 4 nA difference.

from a synthesized circuit to show that simulation and measurement results are well correlated. This level of accuracy is sufficient for use by our optimization engine. Our tool can synthesize netlists for different placement and routing configurations to be simulated by SPICE and can extract metrics of interest from the simulation results. Although using an integrated simulator would increase the tool performance, employing an external simulator brings more flexibility in adopting different simulators as they improve in accuracy and speed.

While routing parasitics mostly affect the performance in digital domain, they may have a serious impact even on the functionality of an analog circuit. Therefore, the emphasis here is mainly placed on optimizing the performance to meet design requirements rather than achieving the most dense packing of the components. During the processes of placement and routing, each circuit net is associated with a different wire type in FPAA. An analog design expert can easily recognize which nets are more sensitive to the routing parasitics than others for a given circuit, and will consider this for making decisions when assigning wire types to each net. The same behavior can be automated by simulating the circuit with different wire types assigned to each net using the interconnect model described above and comparing the performance metrics of interest. If a net is more sensitive to the routing parasitics, metrics obtained by simulating that net assigned to different wire types will deviate more from the metrics of a reference circuit compared to a less sensitive net. A numerical value was extracted from this deviation to generate a net-sensitivity matrix where each entry corresponds to the cost of using a wire type for a net when routing the circuit. In the FPAA with three wire types that was used in this study, the net-sensitivity matrix is of size $3 - by - n$ for a circuit with n nets.

A reference circuit to compare the simulations of different net-wire assignments is necessary for the extraction of the costs in the net-sensitivity matrix. Ideally, one would prefer to use the wires with smallest parasitics for every net, choosing a target circuit configuration as one that uses only local interconnects for every circuit net. This may not be the

case for every circuit since a higher capacitance may sometimes be required for certain nets when FPAA CABs lack large drawn capacitors to satisfy the circuit's requirements. Also, nets connected to the FPAA I/O pins are limited to certain wire types; therefore, they may not be assigned to the smallest capacitance wire type. Considering all these factors, a reference circuit configuration which adds the approximate effects of the non-ideal interconnects while ignoring device and routing constraints is used to obtain target performance metrics that will guide the tool during circuit refinement. In the net-sensitivity matrix, one entry for each net belongs to the target circuit configuration, so these entries must be 0. To find the remaining $2n$ entries, the circuit is simulated with the remaining two wire types of each net while keeping all other nets in the reference circuit configuration, populating the whole matrix in at most $2n + 1$ simulations.

The user may specify the optimization objectives in three directions: equalization, maximization, and minimization. Each direction uses the formulas given below to compute the net-interconnect costs associated with the metrics of interest:

$$Equalize: \quad c_{ijt} = \frac{|X_i - x_{ijt}|}{max(1, |X_i|)}$$

$$Maximize: \quad c_{ijt} = \frac{X_i - x_{ijt}}{max(1, |X_i|)}$$

$$Minimize: \quad c_{ijt} = \frac{x_{ijt} - X_i}{max(1, |X_i|)}$$

where x_{ijt} is the value of metric i when net j is connected to wire type t, X_i is the target value for metric i and c_{ijt} is the interconnect cost for metric i of net j connected to wire type t. If multiple design objectives are specified at the same time, the overall interconnect cost of net j for wire type t (nc_{jt}) can be found as the weighted sum of c_{ijt} for each metric i:

$$nc_{jt} = \sum_{i}^{m} w_i * c_{ijt} \tag{3.1}$$

where w_i is the weight of metric i and m is the number of metrics used for optimization. During the initial placement and refining phases, net-interconnect cost of net j can be found by substituting the value of t, such that $nc_j = nc_{jt}(t)$.

In this work, simulated annealing [68] was used to test the effectiveness of net-interconnect costs in an optimizer. Starting with a valid component-CAB assignment configuration, components are moved into eligible vacancies in other CABs or swapped with components of same type to facilitate new configurations. Our objective function consists of a weighted sum of total switch number (*numsw*), net-wire configuration costs (nc_j) and number of columns spanned by the net ($numcol_j$):

$$cost = \alpha * numsw + \beta * \sum_{j}^{k} (nc_j + numcol_j) \tag{3.2}$$

where k is the number of nets, α and β are constants. In this expression, nc_j reflects the sensitivity of net j to the three wire types while $numcol_j$ reflects the impact of additional columns if type 3 wire is used ($numcol_j$ is 0 for vlwire and vgwire). It is also possible to implement other optimization approaches that exploit the computed net-interconnect costs as long as the device constraints can be adopted into the engine.

3.2.1 Test Suite Setup

Testing the quality of FPAA mapping results has been a difficult issue because of the lack of a realistic circuit suite built using the available component set. Having a variety of circuit classes, each with their own relevant metrics, prevents testing from being a straightforward process. In this work, a test suite with focus on filters was proposed in an attempt to establish a uniform testing criteria. The availability of built-in capacitors and OTAs in FPAA CABs make gmC filters very suitable candidates for this purpose. For this test suite, the focus was only on the ac metrics relevant to important filter specifications. These metrics and their definitions can be found in Table 3.3. Figure 3.14 depicts the physical meaning of each of these metrics on the frequency response plot of a low-pass filter. The objective was observing the impact of incorporating a performance metric into optimization cost function for each circuit. Unlike digital circuits, it is not possible to automate the measurement of a suite of analog circuits and the manual measurement process is prohibitive as the number of circuits and experiments grow; therefore, only the simulation results were included in

this work relying on the correlation between measurement and simulation results presented earlier in [74].

Table 3.3. Performance metrics of a low-pass filter and their definitions.

metric	definition	optimization objective
`fc`	cut-off frequency [Hz]	equalize to target
`gpass`	pass-band gain [dB]	equalize to target
`rp`	pass-band ripple [dB]	minimize
`ror`	roll-off rate [dB per decade]	maximize

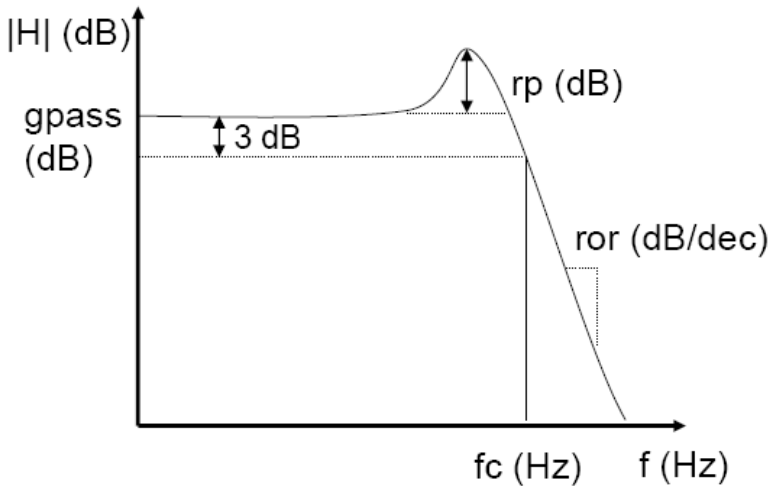


Figure 3.14. Performance metrics of a low-pass filter.

A gmC filter can be systematically built by adjusting the transconductance (gm) of the OTA using state-space equations of any desired filter specifications [75]. MATLAB was used to generate state-space equations and SPICE netlists for 5 types of low-pass filters including butterworth, chebyshev, inverse chebyshev, bessel and elliptic. Description and average synthesis time (including simulations, place & route and optimization for the tested metrics) of all circuits are given in Table 3.4.

3.2.2 Optimization Results

The test circuits described in Table 3.4 were placed and routed using RASPER with WinSpice3 [76] support running on a Pentium4 2.4 GHz machine with 1 GB RAM and Windows XP operating system to collect performance metrics listed in Table 3.3 for different

Table 3.4. Test bench circuits.

filter name	filter type	order	#cmp	time(s)
b_lp8	butterworth LP	8	16	7.7
bs_lp7	bessel LP	7	16	7.4
c1_lp5	chebyshev LP	5	12	5.5
c2_lp5	inverse chebyshev LP	5	19	13.3
e_lp7	elliptic LP	7	32	34.0

optimization goals. These goals include minimization of switch number (*switch_opt*) or `rp` (*rp_opt*), equalizing `gpass` (*gpass_opt*) or `fc` (*fc_opt*) to the target performance metrics, and maximizing `ror` (*ror_opt*). For each optimization goal based on a performance metric, that metric was inspected individually in the net sensitivity cost extraction. Performance metrics of circuits for each of these optimization goals along with the initial placement solution (*pre_opt*) were obtained and presented in Figures 3.15, 3.16, 3.17, and 3.18. In Figure 3.15 the relative errors of the resulting `fc` values from the target `fc` values were demonstrated. All obtained `fc` values in Figure 3.15 are within 2.5 to 4.5 kHz range. Since target `gpass` values are close to 0 dB, demonstrating the absolute errors rather than the relative errors was preferred for `gpass` in Figure 3.16. Figures 3.17 and 3.18 present the `rp` and `ror` values. Since optimization objective is maximization for `ror`, higher values mean success. For all other metrics, lower results or lower errors with respect to the target values are desired.

Using an analog circuit simulator in an optimization loop that requires multiple iterations is prohibitively time consuming. In this work, the simulator was replaced with numerical net sensitivity costs to complete the optimization task in a feasible time while pursuing the performance goals set by the user. In a digital circuit reducing the critical path length, which corresponds to minimizing the number of switches in our case would be a good approach to reduce the circuit delay. When the results presented in Figures 3.15, 3.16, 3.17, and 3.18 are inspected, one can see that switch number minimization alone can actually degrade the circuit performance whereas using net sensitivity costs extracted from

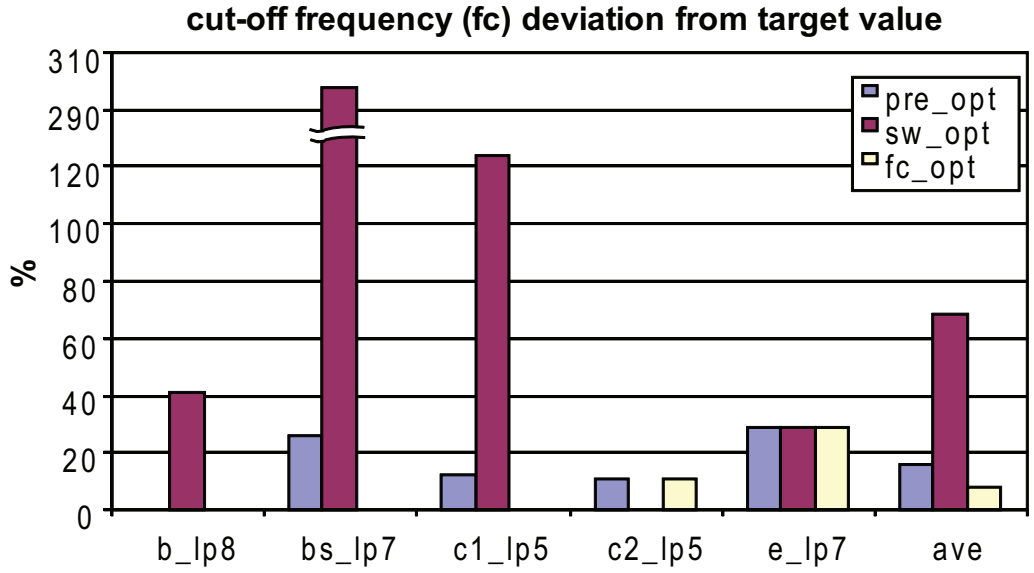


Figure 3.15. Cut-off frequency (`fc`) errors for different optimization goals (lower better, *fc_opt* gives the lowest average values). Missing bars equal 0.

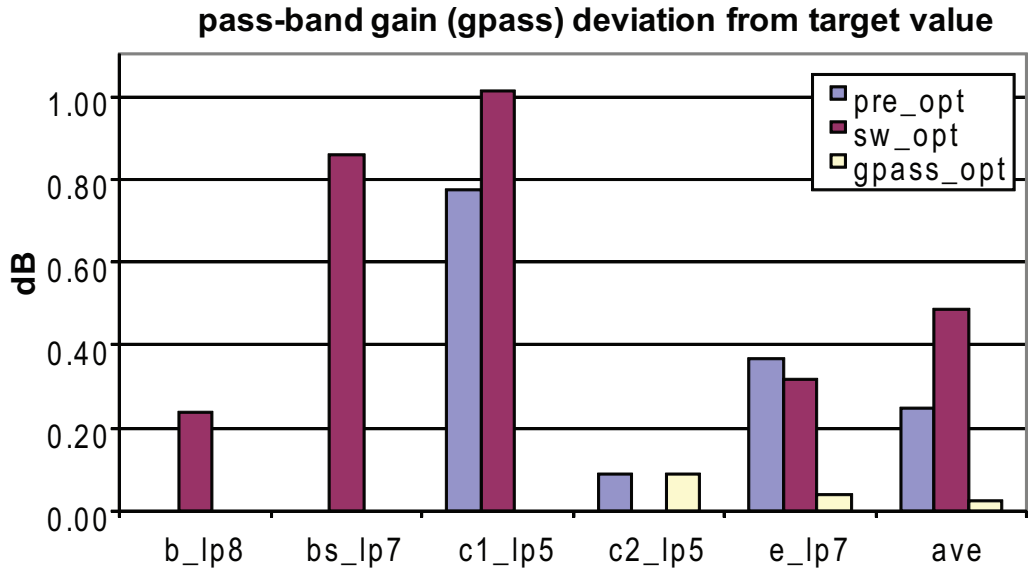


Figure 3.16. Pass-band gain (`gpass`) errors for different optimization goals (lower better, *gpass_opt* gives the lowest average values). Missing bars equal 0.

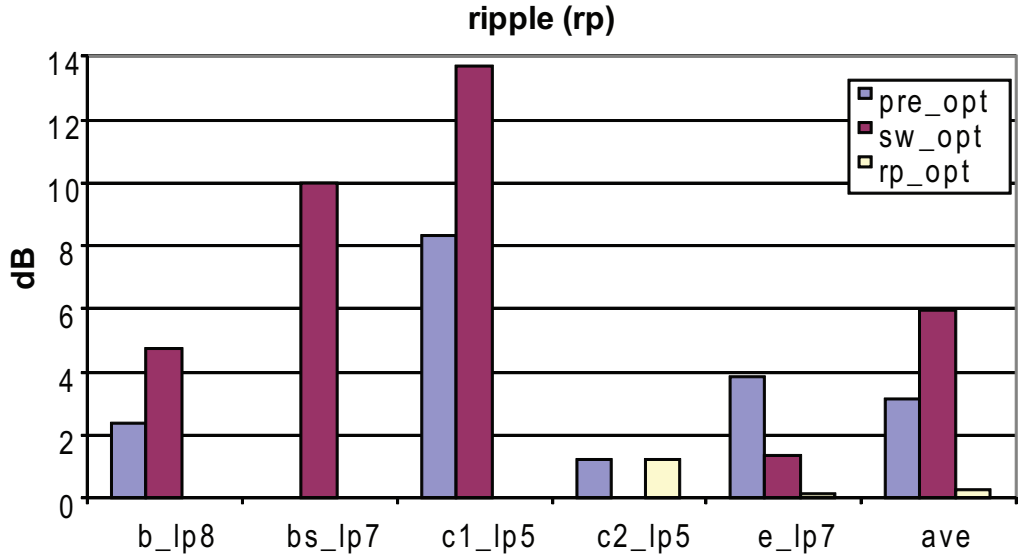


Figure 3.17. Pass-band ripple (`rp`) values for different optimization goals (lower better, *rp_opt* gives the lowest average values). Missing bars equal 0.

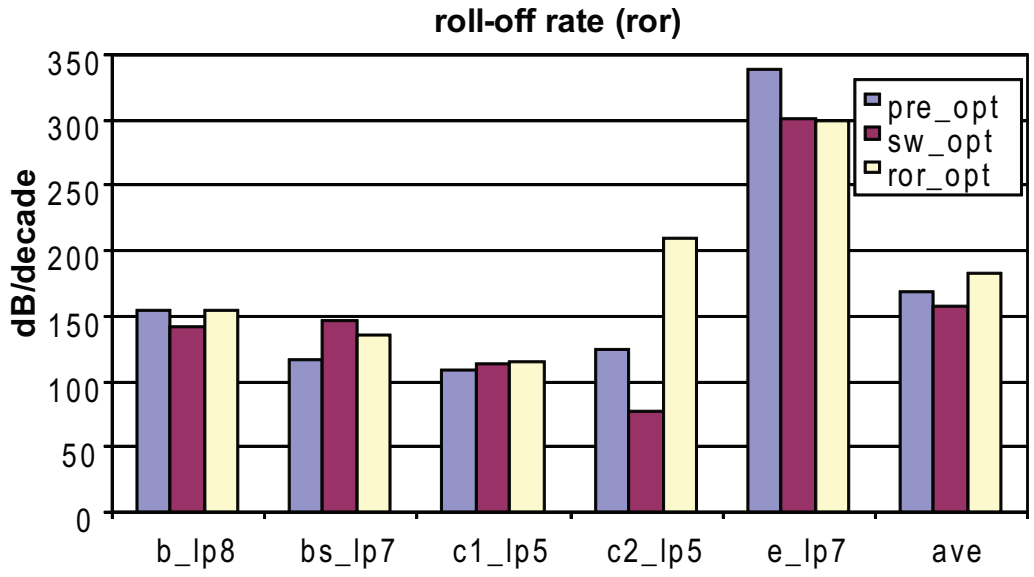


Figure 3.18. Roll-off rate (`ror`) values for different optimization goals (higher better, *ror_opt* gives the highest average values).

performance metrics results in better or similar performance compared to the initial circuit or switch minimization alone. Figure 3.15 reveals a 50% reduction by using *fc_opt* compared to *pre_opt* for `fc` error from the target `fc` value. Likewise, a 90% error reduction was observed for `gpass` in Figure 3.16, a 91% decrease in `rp` value was observed in Figure 3.17, and a 9% increase in `ror` value was observed in Figure 3.18 when net sensitivity costs based on these metrics were used. The only case where switch minimization seems to work slightly better is c2_lp5 (inverse chebyshev filter), which is a robust circuit with acceptable performance metrics for all optimization methods.

In these experiments, only one performance metric is used to form the cost function for each optimization method. Optimizing multiple metrics simultaneously requires special care, since some metrics will be conflicting with each other and it won't be possible to optimize all at the same time. For instance, it is very difficult to increase the roll-off rate and decrease the ripple of a circuit at the same time. Also, since each metric have different numerical value ranges, their relative weights should be assigned carefully.

3.3 Parasitic Extraction in GRASPER

Since the available wiretypes are not predefined but can be generically described in GRASPER, the task of parasitic extraction becomes more challenging. Subcircuits for different wires on the routing path are not readily available and have to be created as part of the task. This process requires extra effort, but also helps capture the interconnect structure more realistically.

GRASPER also uses subcircuit blocks that are inserted between the component pins in a similar fashion to RASPER to simulate the effects of interconnect on circuit performance. Two basic blocks are used for capturing the effect of wires and switches on the interconnect; namely, switches and wire segments.

Switch blocks are two-port subcircuits intended to simulate the effects of the programmed switches only, off switches are represented by a capacitance for simplicity. SPICE

models for the on switches can be acquired from the technology files for the given architecture. This allows customization of the model for desired level of accuracy as in the case of RASPER. For a higher level of accuracy, an active device such as a transistor-based model can be preferred. For a low level of complexity, the subcircuit can consist of an equivalent impedance that approximates the voltage-current relationship of the switch when it is on.

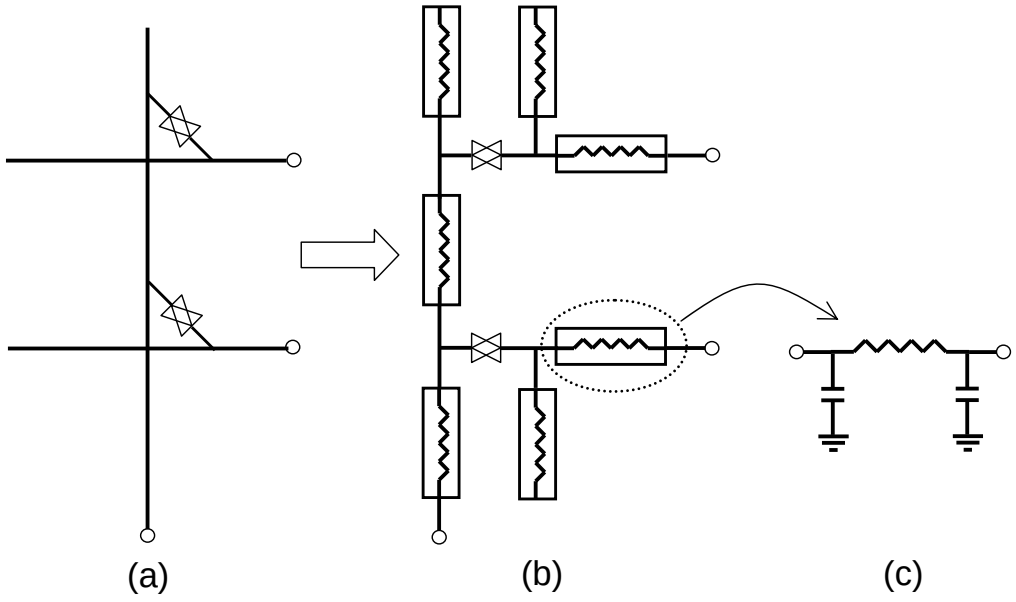


Figure 3.19. Extraction of wire segments from wires and switch locations in GRASPER. (a) Switches that connect wires divide each wire to different segments. (b) Subcircuits are generated for each wire segment. (c) Each wire segment is essentially an RC chain.

Wire segments are also two-port subcircuits generated by inspecting the wire and switch characteristics on a portion of wire that is bounded by the programmed switch locations. At each programmed switch location, wires are divided into segments that can be connected to other wire segments or switches at one or both ends, but not anywhere in between. Figure 3.19 illustrates the process of generating wire segments from the wires and locations of the switches that divide them. Subcircuit descriptions for each wire segment are generated exclusively using information from the architecture configuration file such as segment length, total number of switches on the segment, wire resistivity, wire capacitance, and off switch capacitances. GRASPER can recognize grids as the minimum distance allowed between

any two switches on a wire; and each wire segment is modeled as a linear chain of smallest wire pieces between two such grids that is called as a wire grid. Each wire grid is modeled as a resistive element between ports of the wire grid and two capacitive elements between each port and ground. In this model, the total wire resistance and capacitance is distributed to the smallest wire units, unlike the lumped impedance model that was previously used in RASPER parasitic extraction. Distributed impedance models are more accurate and reliable than lumped impedance models, especially for higher frequencies of interest. After the addition of the off switches to the grids they exist, a distributed RC ladder is generated for each wire segment as illustrated in Figure 3.20. Note that off switch capacitances are added only at the grids that actually contains a switch as described in the architecture configuration file.

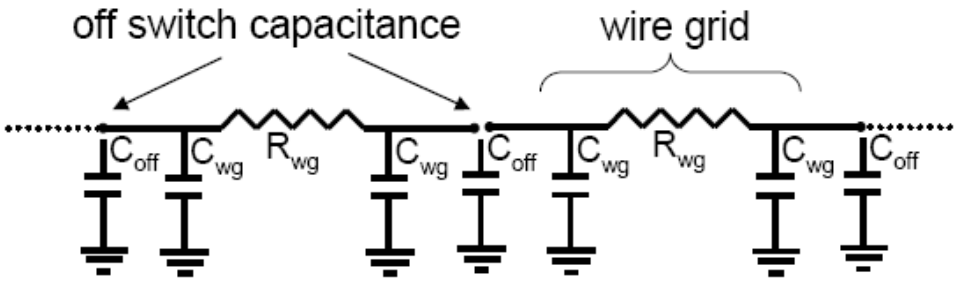


Figure 3.20. Parasitic extraction in GRASPER. Rather than extracting a lumped capacitance and resistance for the whole wire segment, resistances and capacitances of wire grids are added into the chain, distributing the RC components to the whole wire segment.

After the extraction of all wire segments in the interconnect, these wire segments are connected to each other directly or via subcircuit blocks of the programmed switches. SPICE models for I/O pads from the technology file are also added into the interconnect subcircuit where applicable. To reduce the number of subcircuit descriptions, all blocks within the entire interconnect is flattened, so that only one subcircuit description per circuit net is sufficient. The number of ports of this subcircuit is determined by the number of component pins connected to the circuit net associated with the interconnect.

Inductances are not included in this implementation of GRASPER parasitic extraction. They can be included in a future revision to account for the potential RF support of future

reconfigurable architectures.

3.3.1 Model Order Reduction Using the Time Constant Equilibration Reduction Method

Since parasitic component extraction starts from the level of smallest units of wire, the resulting interconnect subcircuit can contain an overwhelming number of parasitic components. Although this may be necessary for simulation accuracy over a wide range of frequencies of operation, it may be an overkill for the frequencies of interest as well. Model order reduction techniques provide with effective ways to reduce circuit complexity while preserving the model fidelity. Sheehan proposes a method for deciding the nets of a circuit that can be eliminated for a minimum time constant of interest in TICER [77]. In this method, time constant for each net can be individually computed using the impedances of components, which are referred to as branches, that connect this net to other nets. TICER finds slow nets and quick nets for eliminating the nets below and above the frequencies of interest respectively. A net is a quick net if its time constant is smaller than the minimum time constant of interest. Similarly, a slow net has a time constant higher than the maximum time constant of interest. GRASPER eliminates only the quick nets, since slow net elimination does not offer any reduction in the complexity of extracted interconnects for reasonable values of maximum time constant. A net's time constant is computed as

$$\tau = \frac{\sum c_i}{\sum g_i} \tag{3.3}$$

where g_i is the conductance and c_i is the capacitance of each branch i connected to the net. It should be noted that the term net used in this context refers to the intermediate signal nodes within the interconnect network and should not be confused with the actual circuit nets that these interconnect networks are generated for.

A quick net can be eliminated by removing all the branches connected to the quick net and inserting new branches between every pair of other nets connected to the removed branches. The impedance of the new branch inserted between two neighbor nets i, j of the

eliminated quick net can be approximated as

$$E_{ij} \approx \frac{g_i g_j}{\sum g} + s\frac{g_i c_j + c_i g_j}{\sum g} \tag{3.4}$$

where $\sum g$ is the total conductance of all branches connected to the quick net, $g_{i,j}$ are the branch i,j conductances, and $c_{i,j}$ are the branch i,j capacitances [77].

The quick net elimination method described above allows removal of components connected to the quick net at the cost of new components created between every pair of neighboring nets. If the quick net has two neighbors, only one components is added while two are being removed, reducing the component count by one. When the quick net has three neighbors, there is no reduction in the number of components. For quick nets with neighbors more than three, quick net elimination increases the number of components; therefore, is not useful for the objective of reducing the model order.

But even eliminating two or three-branch nets can be quite useful when used on the regular RC chains that build the wire segments. As illustrated in Figure 3.21a, the resistor is simply taken out of the network and the capacitor is connected to the next net when a two-branch net is eliminated. Since wire segments are regular chains of resistors and capacitors, another capacitor connected from the neighbor net to ground will be in parallel with the first capacitor, and can be reduced to a single capacitor by finding their parallel equivalent. Then, the next net also becomes a two-branch net and can be reduced in similar fashion if it has a time constant below the minimum time constant. If two-branch nets can not be reduced because of the increased capacitance on the net, net elimination can continue with the three-branch nets as shown in Figure 3.21b supported by parallel reduction of capacitors of the resulting Δ-networks.

Minimum time constant corresponds to the maximum angular frequency; i.e., $2\pi * (maximum frequency)$. Having the option to decide the minimum time constant value gives control of adjusting the parasitic circuit accuracy and complexity to the user.

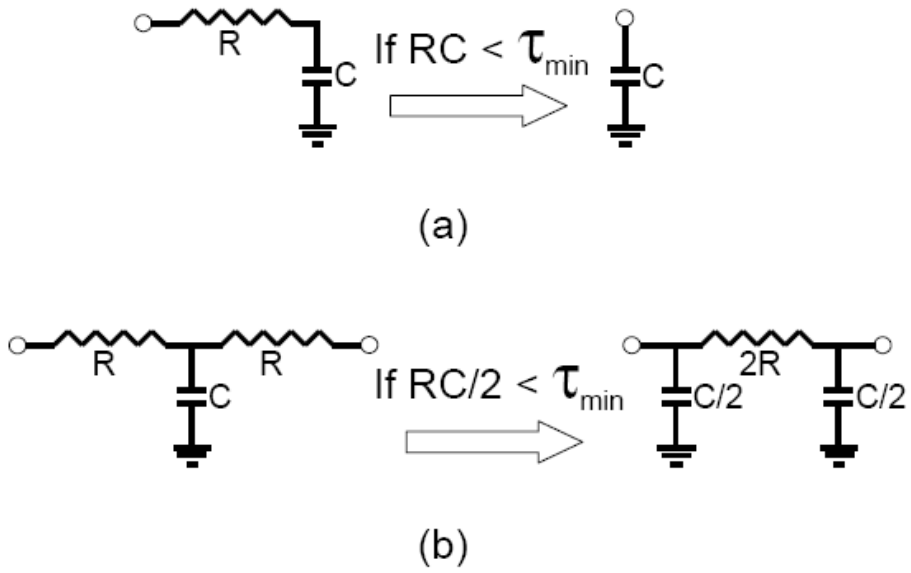


Figure 3.21. Reducing the number of parasitic components in arrangements that are most commonly encountered in interconnects extracted by GRASPER. Using the technique (a) on a net that has one resistor and one capacitor as branches effectively removes the resistor, (b) whereas the number of components of a Y-network of two resistors and one capacitor will be rearranged in a way that allows parallel reduction of capacitors of the resulting Δ-network with the capacitors of potential neighbor networks.

3.3.2 GRASPER Parasitic Extraction Results

One of the goals in developing GRASPER was to make it independent of a particular architecture, so that different FPAA architectures exhibiting variations in their wire and CAB topologies can be supported in the future without major updates to the tool. Therefore, 5000 architectures similar to RASP2.8 [65] were tested by varying the number of components and wires, switch matrix densities, and wire segmentation levels. An 8th order butterworth GmC filter was placed and routed on each of these architectures. Parasitic impedances were also extracted to observe the impact of placement and routing. Among these 5000 architectures, 4747 (94.9%) were routed successfully and 4715 (94.3%) demonstrated low-pass filtering functionality with a reasonable deviation from the target frequency response. Simulation results for this experiment are presented in Table 3.5.

GRASPER placement and routing results were also tested on an actual circuit and FPAA, using a RASP2.8 FPAA evaluation board powered by HP E3610A power supply,

Table 3.5. Performance metrics for an 8th order low-pass filter before and after routing on 5000 FPAA architectures.

metric	ideal	mean	std
cut-off frequency (fc) [Hz]	9976	10939	1363
pass-band gain (gpass) [dB]	0.585	0.878	0.165
pass-band ripple (rp) [dB]	0.362	3.743	1.683

and PCI-DAS 4020/12 scope card to program and measure the response of a 4th order butterworth low-pass filter that is specified using 8 OTAs and 4 capacitors in the circuit netlist. Measurement and simulation results are depicted in Figure 3.22. In addition to the simulation of the input netlist (pre-pnr), two different post-routing simulations (post-pnr sim1 & sim2) are plotted to demonstrate the effect of the simulation models chosen. In the first case (post-pnr sim1) routing switches are approximated by a 10 $k\Omega$ resistor. In the second case (post-pnr sim2), each routing switch is modeled using a floating-gate transistor; resulting in more complex parasitic interconnects. However, the performance metrics summarized in Table 3.6 suggests that this model can simulate the actual circuit performance more accurately.

Table 3.6. Performance metrics obtained from measurements and different simulations of 4th order low-pass filter.

metric	pre-pnr	post-pnr sim1	post-pnr sim2	measurement
fc [Hz]	13804	11641	8222	8500
gpass [dB]	0.097	0.102	0.059	2.5

3.4 TargetC Algorithm for Improved Performance of Routed Circuits

Parasitic capacitances contributed to the circuit by the routing interconnects are major obstacles to meet most design specifications. The logical objective in routing would be minimization of wire and switch capacitances to reduce the impact on the circuit performance.

But if looked into this problem in another way, routing capacitances can also be used to one's advantage, and can even turn a foe into an ally. Capacitors are often used as

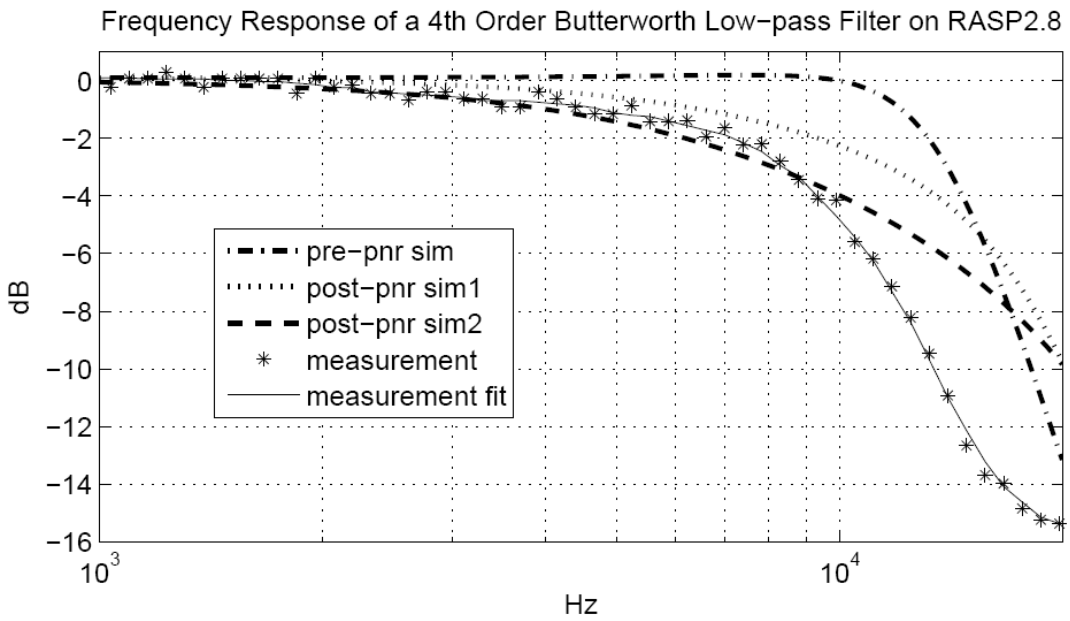


Figure 3.22. Simulations of circuits that correspond to the input netlist (pre-pnr sim), parasitic extracted after routing for different routing switch models (post-pnr sim1 & sim2), and measurement (data points and curve fit) for a 4th order butterworth low-pass filter. Measurements have about 2.5 dB gain, which has been removed here for comparison purposes.

elements of an analog circuit, and RASP architectures contain discrete capacitors in their CABs to be used for this purpose. A capacitor connected to ground can be encountered in many analog designs; RASP2.5 and RASP2.7 architectures even had CAB capacitors connected to ground. Since interconnects also contribute capacitors connected to ground, routing wires can in fact serve as capacitors connected between the routed nets and ground; therefore they can be incorporated into the circuit and eliminate the need for discrete CAB capacitor use. Since CAB capacitors take up a large area, reducing the need for drawn capacitors may relax the active area requirements and the left space can be utilized with other useful components. Most importantly, since the routing capacitances are taken into account to match the target capacitance for each net, design objectives can be met more successfully.

TargetC algorithm is developed for using the routing capacitances to match with the target capacitance for each net. As mentioned in Section 2.2, all components including the CAB capacitors are entered as instantiations of subcircuits in the input netlist. To use

the targetC algorithm, target capacitance values are added into the netlist as constraints for nets that need to have a certain capacitance to ground. This constraint is added as a simple capacitive element line into the netlist in the format:

$$C_i \;\; net_i \;\; 0 \;\; capacitance\_value$$

so that it can be distinguished from the CAB capacitors that are added as instantiations of the corresponding subcircuits. During the circuit graph setup, cells for only the capacitive elements added as subcircuit instantiations of the CAB capacitors are created, and each target capacitance value $tarC_i$ is added as a constraint to the corresponding net i. This style of entering the target capacitances also allows simulation with the same input file.

TargetC algorithm is applied in the routing stage of GRASPER. When each net is routed, capacitance contribution by the routing wires are added up. The total net capacitance $netC_i$ is the sum of all routing wire capacitances for net i and discrete CAB capacitors, if any, connected to net i. Then a discrete capacitor of capacitance C_d is added to net i if $netC_i$ is below a threshold capacitance value $maxC_i$ such that

$$maxC_i = tarC_i - \frac{C_d}{2} \tag{3.5}$$

After the addition of the discrete capacitor, $netC_i$ is evaluated again. Discrete capacitors are added to net i while $netC_i$ is less than $maxC_i$.

Table 3.7. Comparison of cut-off frequencies (fc) of the circuits with ideal interconnects, parasitic effects of routing, and parasitic effects of routing with targetC algorithm.

	ideal	routed	targetC	routed error	targetC error
blp8	9976	6745	10209	32.39 %	2.34 %
c1lp7	7924	5610	7568	29.20 %	4.49 %
c2lp5	8298	5741	8491	30.81 %	2.33 %
elp4	9527	6295	11194	33.92 %	17.50 %

Four low-pass gmC filters have been routed and simulated using this algorithm. The results are reported in Table 3.7. From these results, one can see the effectiveness of the targetC algorithm in bringing the cut-off frequency closer to the target value.

CHAPTER 4

FPAA ARCHITECTURE EXPLORATION

Availability of CAD tools can accelerate the design process by orders of magnitude. As users are able to reach the limits of using the existing reconfigurable hardware, IC designers feel the pressure of improving and expanding their designs. Developing new reconfigurable array architectures, like developing any other hardware system, requires making many strategic decisions. Under the conditions defined by scarcity of the resources, the architectural choices made in the system design process directly influence the usability and performance of the product; therefore, should be approached carefully.

4.1 Previous Studies on FPGA Architecture Exploration

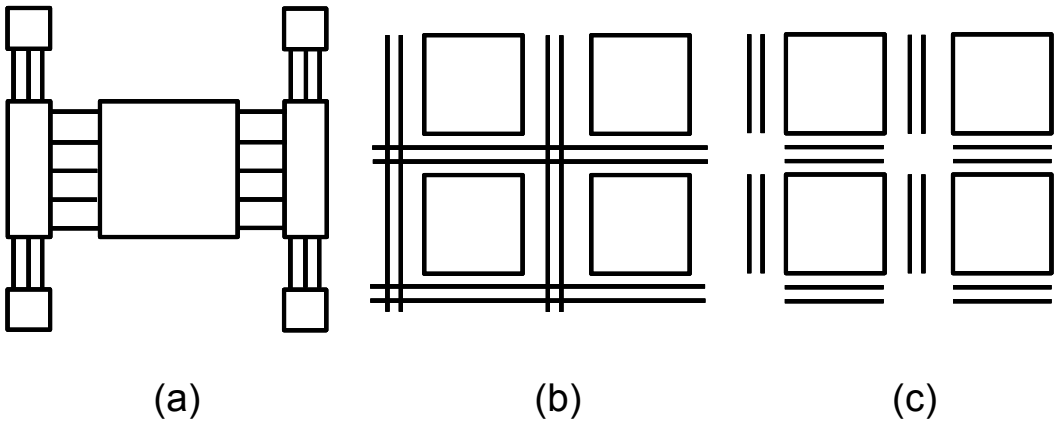


Figure 4.1. Some interconnect topologies that can be used for configuring the connections: (a) fat tree (hierarchical), (b) bus, (c) mesh (nearest neighbor).

The necessity for making high level design decisions wisely leads to another interesting area of research; namely, design space exploration. Many studies have been done in the past to explore FPGA [78, 79, 80, 81, 82, 83, 84, 85, 86] and microprocessor [87, 88, 89] architectures. To decide the appropriate design (input) and solution (output) spaces for FPAA architectures, looking into how previous FPGA architecture exploration research has been done can be useful. Architecture decision categories for design space exploration

in FPGA can be mainly summarized as the following:

- Interconnect topologies: Figure 4.1 illustrates some of the topologies that can be used for interconnect organization that can be counted as fat tree (hierarchical), bus, and mesh (nearest neighbor) connections [88].
- Segmentation levels: Routing wires connecting the computational logic blocks (CLB) may span different number of blocks.
- Switch types: Switch fabric can be constructed using various switch technologies such as pass-gates or tri-state switches.
- Functional units: The way logic is implemented, among the choices of look-up tables (LUT), programmable logic gates, multiplexers, and And-Or gates, can influence the quality of outcome.
- Memory and register resources.

The following metrics are the most commonly measured responses for the explored FPGA architectures:

- Total power consumption
- Total area
- Utilization (for both active and routing area)
- Performance (delay, maximum frequency, etc.)

4.2 FPAA Architecture Exploration

Some of these categories can find applicability into FPAA architecture exploration as well. Results of varying interconnect topologies, segmentation levels, and functional units altogether can reveal possible interaction between these input factors. Switch types are not

likely to interact with the other factors, since they dont affect the placement and routing results for a given technology mapped circuit, but only influence the circuit performance through their parasitic contribution; therefore, exploring their impact on the architectures in a separate study will be more appropriate. Memory and register resources are inapplicable to FPAAs.

The interconnect topology of RASP2.8 can be considered as a hybrid of bus and mesh interconnects. To limit the problem size, only similar interconnect topologies are investigated in this work. In this topology, CABs are organized as a regular rectangular array, and there are horizontal global (hg) and vertical global (vg) wires that span all the columns or rows of CABs. In addition, there are vertical wires with smaller span covering only a portion of all CAB rows, starting from vertical wires of span 1 (vertical local wires). These wires are supported by the horizontal nearest neighbor (hn) wires, which are in fact vertical wires that establish connection between two CABs that are neighbors in the horizontal orientation. The wire types to observe the effects of different levels of segmentation are illustrated in Figure 4.2.

In FPAA, while increasing the availability of routing resources can improve the efficiency of routing, it can also result in the increase of parasitic effects on the circuit. A previous study for the FPGAs shows that increasing the number of routing resources may not really be necessary for improving the routability of the target circuits [90]. Since the effects of routing parasitics can be very critical to the performance of the circuits that will be implemented on the FPAA, it is worth investigating the tradeoffs between the ease (i.e. number of wires and switches) and the quality (i.e. circuit performance) of routing. Adding the number of each wire type to be explored and the switch matrix density into the input space takes these factors into account. Switch matrix density is the ratio of the number of physical switches available to the total number of possible switch locations on the switch matrix. Having a full switch matrix increases the routability at the cost of adding extra capacitance on the wires they are connected to. Finally, the number and distribution of the

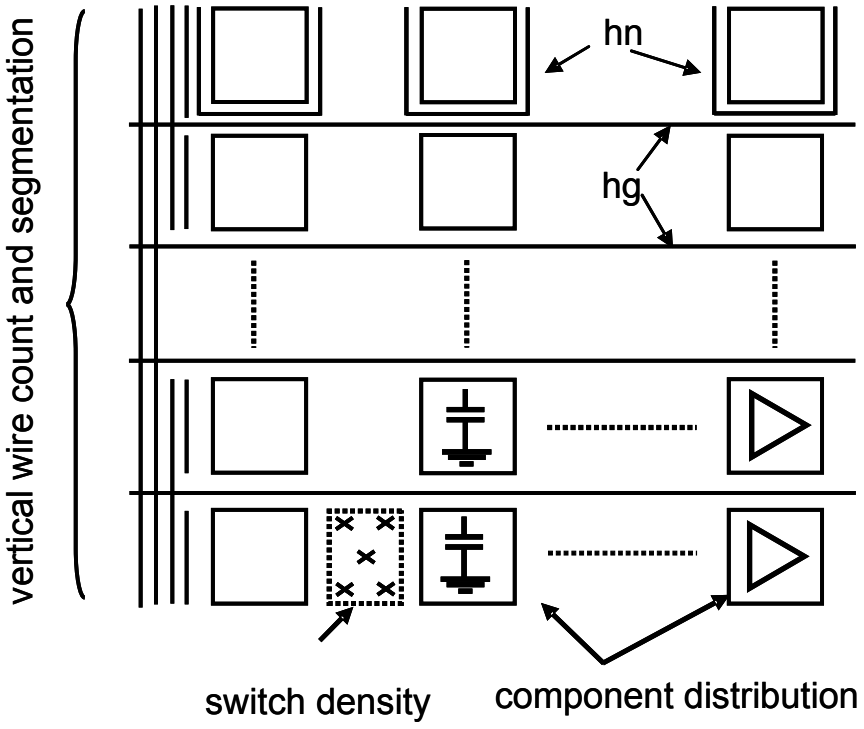


Figure 4.2. Variation of wires with respect to their orientations and span (segmentation).

functional units have a direct influence on the success of routing. The more required units each CAB has, the more likely it is to find local routing solutions, but at the same time device utilization and wire parasitics will be affected by the increased CAB size and CAB pin number.

The power of GRASPERs features can be appreciated better in the study of architecture exploration. Since no fixed architecture is hardcoded into GRASPER so that it can be configured to support many architectures, it provides the ability to collect placement and routing statistics for many FPAA topologies within the design space to be explored. GRASPER also has a parasitic extractor, which can be used to simulate the effects of routing parasitics on the circuit performance by means of an external simulator such as SPICE. Considering the number of possible architectures to test, fabrication and measurement is not a feasible option. Accuracy of the simulation models is useful; however, since the objective is capturing the trends of the design parameter variation effects, fidelity of the simulation is more important and necessary.

GRASPER provides a powerful tool for evaluating different architectures efficiently. But, how can it be used effectively to explore a design space and find the optimal architecture among many design points? It is also worth noting that each circuit class may dictate different demands resulting in a different optimal architecture; therefore, the target circuit factor has to be considered as well. When the combinations of all possible values of different input parameters in a design are considered, the number of design points to test can be overwhelming. This work uses the design of experiments approach that is reviewed in Section 4.3 for systematically exploring the design space to find an optimum architecture for each class of circuits. The results of the case studies in this work are expected to give useful feedback to the hardware designers. But more importantly, the methodology developed here can also be considered as an indispensable part of the reconfigurable architecture design process.

4.3 Design of Experiments

Design of experiments (DOE) is a systematic approach to efficiently characterize and optimize a process by collecting and analyzing data from experiments that may or may not be controlled by the experimenter [91, 92]. In the case where experimenter has no control over the process, the outcome of each experiment is conventionally called an observation. The factors that affect the outcomes of the experiment are referred to as input factors, and the particular results the experimenter is seeking in the outcomes are called as responses. Varying only one input factor at a time gives the experimenter confidence when determining the effects of each input factor separately at the cost of increased amount of resources and time used. This method rapidly becomes infeasible as the number of factors and the levels they can take increase. Statistically designed experiments, on the other hand, help determine the impact of multiple input factors simultaneously with less effort.

DOE is an effective approach that has been used widely in the fields of agriculture, mining, environmental sciences, biomedicine, and manufacturing to analyze statistical data

for drawing conclusions or to design experiments efficiently. DOE methodology consists of the stages below:

- Screening of input factors to verify they actually have an effect on the responses.
- Planning and conducting the experiment.
- Fitting models to the experimental data.
- Optimization or conclusion based on the obtained models.

There are several approaches to designing an experiment. These approaches can be classified into the following three groups:

- Classical designs
- Optimal designs
- Space-filling designs

Among the classical designs, one can find full factorial, fractional factorial, central composite, and robust designs as the most commonly used design types. All possible input factor level combinations are included in a full factorial design, which results in $\prod_i k_i$ design points where k_i is the number of levels for input factor i. For the common case where each factor is divided into two levels, number of design points is 2^n. This design can support high level interactions between the input factors; however, the number of points grow so fast for increasing number of levels and factors that it is not practical for most cases. A 2^{k-p} fractional factorial design helps reduce the number of points in such a case by selecting a $\frac{1}{2^p}$ fraction of the corresponding full factorial design. Fractioning the design results in confounding the interaction effects between some of the input factors; i.e., these effects can not be separated from each other. Central composite designs are obtained by adding two axial points for each input factor and a central point (2k+1 points total) to

an existing full or fractional factorial design. An illustration of these three designs for 3 factors with 2 levels is presented in Figure 4.3. Classical designs are well researched and very effective for relatively simple regions; however, they are not very suitable for covering a wide range of inputs.

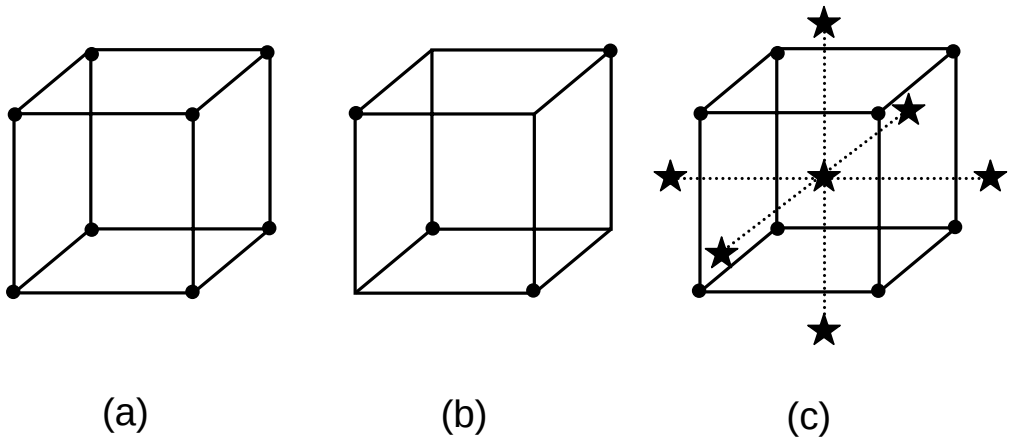


Figure 4.3. Classical experimental designs for 3 factors with 2 levels: a) full factorial design, b) fractional factorial design, c) central composite design.

Optimal designs are most useful when the system is relatively well-known, and there is some confidence on the most suitable model for this system. Using this style is also suitable when there are known constraints on the points of a classical design, such as certain corners of a factorial design being infeasible. Essentially, the experimenter already has an idea about what model to use, and trying to optimize the design by picking points at the right places that will minimize the error for that particular model.

Using space-filling designs is preferable when there is not much information about the system and the appropriate model. In space-filling designs, the overall design space is regularly or randomly sampled for a fixed number of design points of the experimenters choice. Increasing the number of levels for input factors multiply the design points in a classical design, whereas in the case of space-filling number of design points do not have to increase as much. This property makes space-filling designs a better choice for maximum coverage of wider input ranges, and allows capturing global models of the system. There

are variants of space-filling designs with respect to the sampling method used. In random sampling, each input is randomly selected from the range of values for that input factor. This randomness does not guarantee equal representation of different portions of the sample space; therefore, in a second approach called stratified sampling the entire range is spread evenly into n strata, and points are randomly selected equally from each stratum. Finally, a third approach called Latin Hypercube Sampling (LHS), divides the sample space for each input factor into N strata of equal marginal probability $\frac{1}{N}$ where N is the number of samples, and takes samples only once from each stratum. LHS is essentially a high dimensional generalization of Latin square sampling, and results in better representation of the real variability [93, 94]. In this work, space-filling design with LHS was used for designing the experiments and collect data for FPAA architecture exploration. Figure 4.4 depicts a 3-dimensional view of an experimental design using LHS.

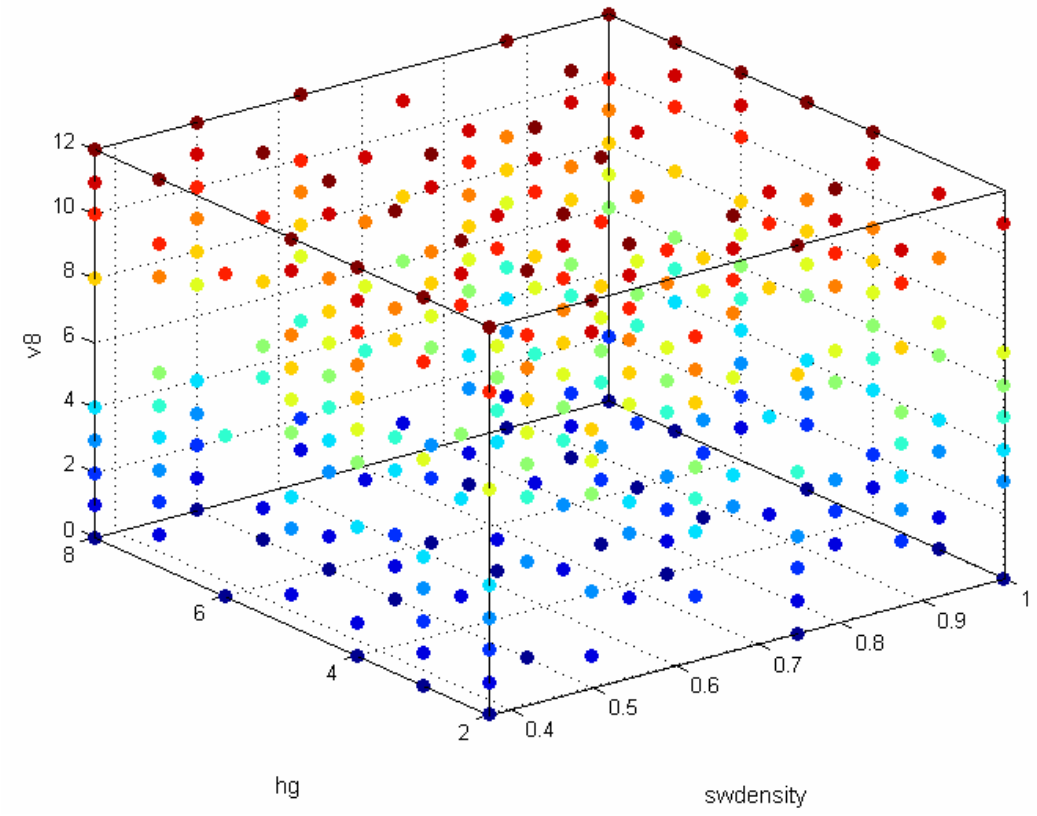


Figure 4.4. A 3-dimensional view of an experimental design using latin hypercube sampling (LHS).

4.4 Fitting Polynomial Regression Models to Experimental Data

The next step after collecting the experimental data is finding suitable models to approximate each response for different values of input factors. The main purpose in modeling is being able to predict the responses for any point within the entire design region without having to execute the experiment for all those points. A simple approach to obtain such a model is fitting a multi-variable polynomial that accepts input factors of the design as its variables using regression techniques. Figure 4.5 visualizes the response surface formed when such a polynomial regression model is fit to the design data.

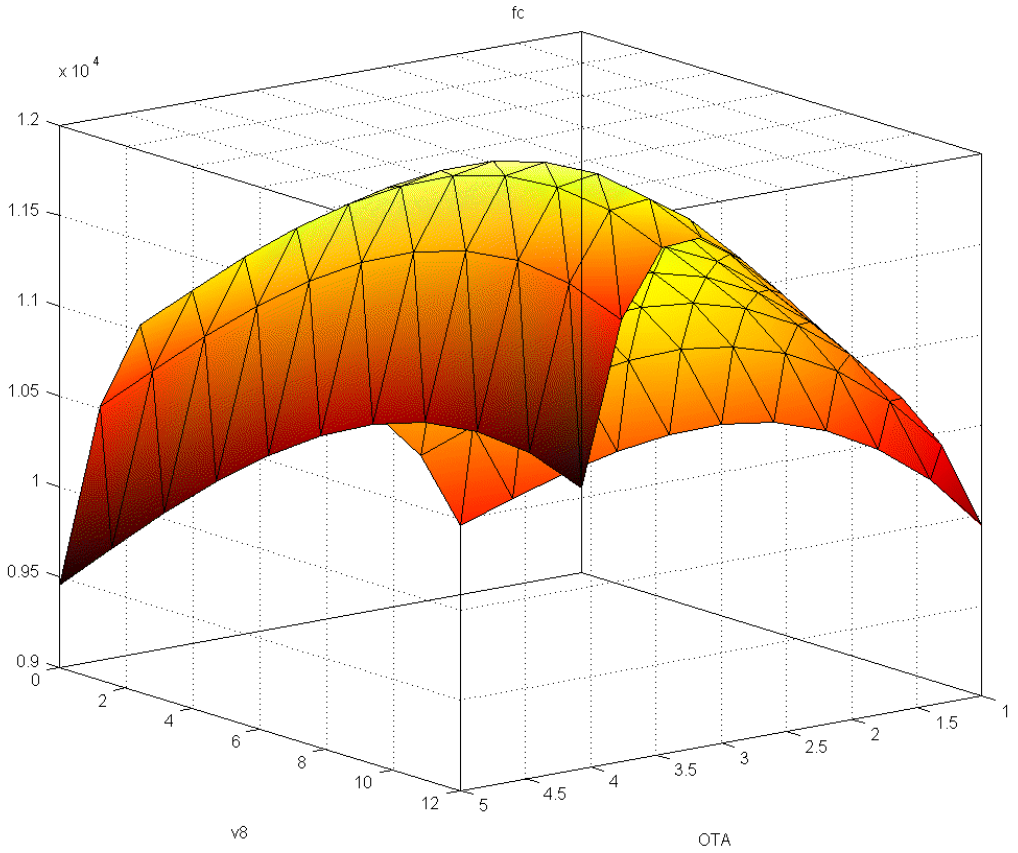


Figure 4.5. The response surface in a polynomial regression model.

A polynomial regression model is a Taylor series expansion with a finite number of polynomial coefficients. These coefficients may consist of a constant term, different order terms for each factor, and interaction terms. The lower polynomial orders of one, two, and three are commonly referred to as linear, quadratic, and cubic respectively. Linear

terms in the polynomial reflect the main effects of each input factor, while nonlinear terms can be visualized as curvatures in the response surface. If there are dependencies between two or more input factors, these dependencies are represented in the interaction terms of the polynomials. The order of a polynomial regression model in each variable is determined by the number of distinct points for the corresponding input factor among the design points. Therefore, if number of levels is k_i for input factor i, the maximum polynomial order in variable i can be $k_i - 1$. Equation 4.1 gives an example to a quadratic (second order) polynomial regression model with interaction order 2:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_2 + \beta_4 x_2^2 + \beta_5 x_1 x_2 + \varepsilon \tag{4.1}$$

In the polynomial model given above, x_i are called the predictor or input variables, β are estimates (of the polynomial coefficients), and y is the response that is being modeled. When an input vector $\vec{X}$ is evaluated using this model, the result is called the regression predictor of y, and conventionally shown as $\hat{y}$:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_1^2 + \hat{\beta}_3 x_2 + \hat{\beta}_4 x_2^2 + \hat{\beta}_5 x_1 x_2 \tag{4.2}$$

where $\hat{\beta}$ are estimated through ordinary least squares. The error ε between the actual response value y and the predicted response $\hat{y}$ is called the residual.

To determine the coefficients in a polynomial regression model, least squares estimates method is used. Least squares estimates method is based on minimizing the sum of squares of the residuals, which is also called as sum squared errors (SSE), for the set of training data used in fitting a model. Inputs in the data set may be mapped to [-1, 1] range to prevent any factor dominating over the other factors due to different ranges of numerical values. A successful model fit is one that has a small SSE and a normal distribution of residuals with mean 0. Unfortunately, since the value of SSE depends on the range of response values, there is no fixed threshold to decide what SSE values reveal a successful fit. To appreciate a small SSE value, it is best to evaluate it with the total sum of squares as well. Total sum

of squares (SST) is directly obtained from the response values in data set as:

$$SST = \sum_{i}^{n} (y_i - \overline{y})^2 \tag{4.3}$$

where y_i is the observations of response and $\overline{y}$ is the mean response value. Then, another metric called as the coefficient of determination, also referred to as R^2, can be calculated as:

$$R^2 = 1 - \frac{SSE}{SST} \tag{4.4}$$

taking values between 0 and 1. When R^2 approaches 0, the residuals obtained using the regression model do not display any better performance than using the sample mean as estimated response for all inputs. An R^2 value close to 1 is a good sign of a successful model fit.

Without sufficient knowledge of the system, one can find the right polynomial model orders by trial and error. When more terms are included into the regression model with increasing factor orders, R^2 is expected to approach 1 while SSE is reduced. This may be deceiving in some cases, since the model begins to chase the variation in data rather than the trend, resulting in overfitting the data. In this case, the residuals for observations included in the sample data will be smaller, while the prediction ability for the points not included will be lost. To prevent the undesired effects of increasing the model complexity, predicted sum square residuals (PRESS) is used. PRESS is calculated by first removing one point from the observations and fitting the model for the remaining data; then repeating this process for each observation while adding the squares of prediction residuals for the omitted observations. PRESS values can never be less than SSE; however, having PRESS values close to SSE is a good sign that the model is not very sensitive to the sample data, and preserves its prediction ability for the points not tested in the experimental design. Similar to the SSE, a predicted coefficient of determination, also called as R^2_{pred}, can also be derived from PRESS as:

$$R^2_{pred} = 1 - \frac{PRESS}{SST} \tag{4.5}$$

Like R^2, R^2_{pred} also has the maximum value of 1, which tells the model fit is very successful and is likely to predict the responses for points not included in the sample data well. On the other hand, R^2_{pred} can also take negative values when the model prediction results in residuals worse than assuming the response equal to the sample mean for all inputs.

4.5 Systematic Exploration of the Floating-gate-based FPAA

In this work, FPAA architecture exploration was narrowed to bus-style topologies which are similar to the RASP2.8 architecture. Like RASP2.8, the searched architectures are also 8 rows by 4 columns of CABs, connected to each other by vertical and horizontal wires of different spans. In addition, the effects of different switch densities where only a certain proportion of all vertical and horizontal wire intersections are connected via switches, and the number of components included in each CAB were considered. The number of input factors (knobs) was limited to 9 as follows:

1. Switch density (sw): from 0.5 (half connected) to 1.0 (fully connected) with 0.125 increment (5 levels).
2. hgwires (hg): from 2 to 8 (7 levels).
3. v8wires (v8): from 0 to 12 (13 levels).
4. v4wires (v4): from 0 to 12 (13 levels).
5. v2wires (v2): from 0 to 12 (13 levels).
6. v1wires (v1): from 2 to 12 (11 levels).
7. hnwires (hn): from 0 to 4 (5 levels).
8. output transconductance amplifiers(OTA), from 1 to 5 (5 levels).
9. capacitors (cap), from 1 to 5 (5 levels).

These input factors were quickly screened, and their effects on the responses given below were verified. This test consisted of keeping all factors constant at a design point, and varying only one factor to see if the response changes. Since FPAA is a reconfigurable architecture with prefabricated units, all of these inputs must be discrete. Input factor 1 has 0.125 increments; all other input factors take integer values. The 9-dimensional space defined by these input factors with their associated levels contains about 106 million possible points in the design region.

Architecture exploration requires testing the performance of many architectures for a set of circuits. Manufacturing all these architectures is not feasible; therefore, simulations using the parasitic extraction capabilities of GRASPER mentioned in Chapter 3 were inevitable. Even with the simulations, it is necessary to rapidly collect the metrics of interest for a large number of architecture-circuit pairings, strongly necessitating automation. For these reasons, the circuits tested were limited to gmC filters, for which computer software to extract performance metrics such as cut-off frequency (fc), pass-band gain (av), and ripple (rp) was developed. Since gmC filter design requires only OTAs and capacitors, the focus in this study was only on these CAB components. Within the range of architectures specified above, an optimal FPAA for 4 different gmC low-pass filters were searched. These filters are:

1. blp8: 8th order butterworth with 8 capacitors and 17 OTAs.
2. c1lp7: 7th order chebyshev with 7 capacitors and 17 OTAs.
3. c2lp5: 5th order inverse chebyshev with 5 capacitors and 19 OTAs.
4. elp4: 4th order elliptic with 4 capacitors and 17 OTAs.

SPICE netlists for these 4 gmC filters are provided in Appendix C. For each of these circuits, seven different responses with continuous values were modeled. These responses are:

1. Routability: A value that indicates the ratio of successfully routed nets. If routing was completed successfully, routability is 1; otherwise it is a real number between 0 and 1. A feasible architecture is required to achieve a routability value of 1 for all circuits.

2. Cut-off frequency (fc): The frequency at which the gain falls 3 dB below the pass-band gain.

3. Pass-band gain (av): The ratio of output signal to input signal (in dB) at the pass-band region of the filter.

4. Ripple (rp): The difference between the maximum gain over the whole frequency spectrum and the pass-band gain (in dB).

5. Switch utilization (swutil): Percentage of the switches used for routing to the total number of switches.

6. Wire utilization (wireutil): Percentage of the wires used for routing to the total number of wires.

7. Component utilization (cmputil): Percentage of the components used for placement to the total number of components.

Responses 1, 5, 6, and 7 are collected using the placement and routing tool (GRASPER). Responses 2, 3, and 4 are collected by simulating the parasitic extracted SPICE netlist of a successfully routed circuit. The aim is to fit polynomial regression models for each of these responses and use these models together to find the optimal architecture. For this purpose, experiments were designed for the 9 input factors given above. To design the experiments, stratified Latin hypercube with symmetrical points was used. Two experiments were designed to sample different number of design points. The first experiment contains 2000 points and is used for modeling. The second experiment contains 200 points and is

used for validation of the model. Architectures for these design points are generated using automated scripts, and GRASPER is used to place and route each circuit on these architectures, while SPICE is used to collect the performance metrics. The statistics summary for the 2000 data points may be found in Tables 4.1, 4.2, 4.3, and 4.4 to give an idea about the different numerical ranges for each response. In these tables, `min` and `max` refer to the minimum and the maximum values in the data set, `median` refers to the value of data point that is larger than half of the points in the data set, `quartile1` (`quartile3`) is the first (third) quartile value, `mean` is the arithmetic mean, and `stdev` is the standard deviation of data. In the statistics terminology, first (third) quartile is the value of the point that is larger than 25% (75%) of all values in the data set.

Table 4.1. Summary statistics of 2000 data points for the circuit blp8.

	routability	fc	av	rp	swutil	wireutil	cmputil
min	0.275	4886	-0.359	0.047	0.185	6.180	4.427
quartile1	1.000	9310	0.829	1.378	0.443	8.949	5.903
median	1.000	9976	0.892	3.171	0.614	10.384	6.641
quartile2	1.000	10939	0.953	4.546	0.861	12.676	8.854
max	1.000	14420	1.862	9.451	3.987	24.541	13.281
mean	0.989	10081	0.888	3.086	0.701	11.041	7.147
stdev	0.070	1230	0.241	1.951	0.367	2.843	2.110

Table 4.2. Summary statistics of 2000 data points for the circuit c1lp7.

	routability	fc	av	rp	swutil	wireutil	cmputil
min	0.306	4456	-4.089	0.452	0.187	6.271	4.427
quartile1	1.000	7396	5.698	14.824	0.445	8.980	5.460
median	1.000	7924	5.791	17.360	0.616	10.323	6.641
quartile2	1.000	8689	5.908	19.833	0.850	12.380	8.854
max	1.000	11721	11.996	24.119	2.760	23.132	13.281
mean	0.990	7962	5.838	16.511	0.698	10.912	7.145
stdev	0.068	937	1.966	4.602	0.351	2.630	2.110

Table 4.3. Summary statistics of 2000 data points for the circuit c2lp5.

	routability	fc	av	rp	swutil	wireutil	cmputil
min	0.263	3459	-15.316	0.000	0.175	5.980	4.948
quartile1	1.000	8298	0.655	0.201	0.443	8.870	5.938
median	1.000	8891	4.439	0.923	0.625	10.345	7.422
quartile2	1.000	9749	6.648	2.557	0.872	12.327	9.896
max	1.000	32283	13.932	13.438	2.914	24.902	14.844
mean	0.992	9128	3.983	1.737	0.708	10.867	7.972
stdev	0.059	1344	3.998	2.048	0.366	2.736	2.352

Table 4.4. Summary statistics of 2000 data points for the circuit elp4.

	routability	fc	av	rp	swutil	wireutil	cmputil
min	0.444	6591	-18.351	0.000	0.152	5.240	4.427
quartile1	1.000	9749	0.136	0.855	0.386	7.716	5.313
median	1.000	10690	3.254	3.011	0.544	9.005	6.641
quartile2	1.000	11721	4.719	5.124	0.764	11.038	8.854
max	1.000	15451	15.632	15.385	2.528	21.352	13.281
mean	0.996	10669	2.885	3.218	0.622	9.552	7.128
stdev	0.042	1416	3.746	2.419	0.330	2.500	2.098

4.5.1 Selecting optimum orders for multi-variable polynomial models

Various statistical tools are available to aid fitting polynomial regression models for different responses. In this work, model-based calibration (MBC) toolbox of MATLAB was used. This tool requires the experimental data set for the desired response, and a simple description of the polynomial model for which the coefficients will be determined. The polynomial model description includes the maximum polynomial order for each input factor, and the interaction level between different input factors. To achieve more successful models, the objective should be finding the polynomial orders that will result in the highest R^2_{pred}, or the lowest PRESS values possible for the given data. Using the polynomial model fitting functions of MBC toolbox, an algorithm that finds the optimum polynomial models for each response was implemented. The algorithm works as follows: First a sorted search list is determined for the variables by fitting linear polynomials of only one variable at a time, and sorting in ascending order of the PRESS values associated with each variable's

```
Optimum Order Polyfit
1: input: number of poly variables;
2: for (each poly variable)
3:     set order of variable to 1, other variable orders to 0;
4:     compute PRESS for polyfit;
5:     store PRESS value for variable;
6: sort variables in ascending order of their PRESS values;
7: set minError = minimum PRESS value;
8: set all poly variable orders to 0;
9: set max order for each variable = number of input levels - 1;
10:repeat
11:    set minErrorIni = minError;
12:    for (each sorted variable)
13:        while (variable order less than max order and minError is reduced)
14:            increment variable order;
15:            compute PRESS for polyfit;
16:            if (PRESS less than minError)
17:                set minError = PRESS;
18:                set bestpoly = poly variable orders;
19:            else
20:                decrement variable order;
21:until (minErrorIni is equal to minError)
22:return bestpoly;
```

Figure 4.6. Pseudo code for Optimum Order PolyFit.

model. Then, polynomial order of variables from the sorted search list is incremented one by one until PRESS cannot be reduced further for that variable. The same is done for every variable until maximum order for all variables is found in this pass. This process is repeated starting from the first variable in the sorted search list until PRESS cannot be reduced anymore by increasing the polynomial order of any variable. A pseudocode for this algorithm, which is called as Optimum Order Polyfit is presented in Figure 4.6.

As an improvement to the algorithm, one can update the sorted search list after every pass considering the rate of PRESS reduction for each variable in that pass. In addition, a user-defined minimum error reduction value can be introduced to enforce a minimum

improvement in PRESS to accept the increase in the polynomial orders. The latter modification has the advantage of giving the user a control over the complexity of the resulting polynomial models, as well as the time it takes for the algorithm to complete. Using the Optimum Order Polyfit algorithm, and the data collected for four circuits placed and routed on 2000 different architectures, the models summarized in Tables 4.5, 4.6, 4.7, and 4.8 were obtained. In these tables, number of variables included in the model (`variables`), `order` and number of estimates (`coefficients`) of the polynomial model, root mean square (`RMSE`) and predicted root mean square (`PRESS RMSE`) of the regression model, RMSE obtained by testing with validation data (`valid RMSE`), R^2, and R^2_{pred} are provided for each of the 7 responses of 4 circuits. `PRESS RMSE` mentioned above is obtained by dividing the earlier described PRESS statistic into the number of data points, and taking the square root of that value, in a similar fashion to the relationship between `RMSE` and SSE.

Table 4.5. Optimum Order Polyfit results for the circuit blp8 using 2000 design points and 200 validation points.

	routability	fc	av	rp	wireutil	swutil	cmputil
variables	7	9	9	8	9	9	4
order	5	4	5	4	4	4	4
coefficients	58	131	131	75	179	175	17
RMSE	0.050	642	0.193	1.511	0.041	0.364	0.055
PRESS RMSE	0.052	665	0.200	1.539	0.045	0.386	0.055
valid RMSE	0.039	675	0.188	1.563	0.044	0.400	0.056
$R\{2\}$	0.500	0.746	0.403	0.423	0.989	0.985	0.999
$R\{2\}\_\{pred\}$	0.457	0.708	0.311	0.377	0.985	0.982	0.999

When the results in Tables 4.5, 4.6, 4.7, and 4.8 are inspected carefully, one can see very successful, moderately successful, and poor fits depending on the response type and the circuits. Any model related to the utilization responses (swuitl, wireutil, cmputil) is highly predictable as hinted by the R^2_{pred} values hitting very close to 1. The models for fc are fairly decent, especially for the circuits blp8 and c1lp7. The predictive power of the routability models are expected to be less as R^2_{pred} values of 0.5 or less suggest. However, with validation RMSE values ranging from 0.04 to 0.08, these models may still be used by

Table 4.6. Optimum Order Polyfit results for the circuit c1lp7 using 2000 design points and 200 validation points.

	routability	fc	av	rp	wireutil	swutil	cmputil
variables	7	8	4	8	9	9	5
order	6	4	4	6	4	4	4
coefficients	55	83	18	87	176	176	19
RMSE	0.055	534	1.859	3.083	0.034	0.318	0.055
PRESS RMSE	0.057	545	1.868	3.161	0.036	0.338	0.055
valid RMSE	0.081	558	1.598	3.137	0.040	0.351	0.056
$R^{\{}2\}$	0.357	0.689	0.114	0.571	0.992	0.987	0.999
$R^{\{}2\}\_\{pred\}$	0.293	0.661	0.097	0.528	0.989	0.983	0.999

Table 4.7. Optimum Order Polyfit results for the circuit c2lp5 using 2000 design points and 200 validation points.

	routability	fc	av	rp	wireutil	swutil	cmputil
variables	7	8	8	7	9	9	4
order	5	4	4	4	4	4	4
coefficients	83	46	47	40	175	175	14
RMSE	0.046	1109	3.595	1.764	0.039	0.365	0.061
PRESS RMSE	0.048	1121	3.637	1.779	0.041	0.384	0.062
valid RMSE	0.065	1030	3.675	1.818	0.037	0.381	0.061
$R^{\{}2\}$	0.427	0.334	0.210	0.273	0.990	0.984	0.999
$R^{\{}2\}\_\{pred\}$	0.343	0.304	0.172	0.245	0.987	0.980	0.999

Table 4.8. Optimum Order Polyfit results for the circuit elp4 using 2000 design points and 200 validation points.

	routability	fc	av	rp	wireutil	swutil	cmputil
variables	7	8	7	7	9	9	4
order	4	3	5	4	5	4	4
coefficients	73	44	41	39	177	175	14
RMSE	0.036	1182	3.559	2.082	0.035	0.311	0.055
PRESS RMSE	0.038	1193	3.595	2.101	0.038	0.329	0.055
valid RMSE	0.029	1172	3.732	2.147	0.039	0.354	0.055
$R^{\{}2\}$	0.279	0.318	0.116	0.274	0.990	0.986	0.999
$R^{\{}2\}\_\{pred\}$	0.185	0.290	0.078	0.245	0.987	0.983	0.999

allowing a small tolerance of ≈ 0.05; since the highest routability values for the unroutable circuits are observed to be 0.85 for blp8, 0.60 for c1lp7, 0.76 for c2lp5, and 0.79 for elp4. The polynomial model fits for av and rp are not very successful, and it may not be a good idea to strongly depend on their accuracies in the optimization phase. Fortunately, the main requirement in filter design is to keep av and rp within acceptable margins rather than precisely setting values for them.

4.5.2 Optimization Using Polynomial Regression Models

After fitting the polynomial regression models, a vector with its elements as the input factors that define the optimum architecture can be found using the optimization toolbox of MATLAB. There are, however, two challenges in doing so: The first challenge is the requirement to optimize multiple, and possibly competing responses simultaneously. Having to do this multi-objective optimization for more than one circuit only adds to the challenge. The second challenge is that the optimization may result in a vector with real numbers as its elements because of the limitations of MATLAB in enforcing discrete constraints; hence, cannot be implemented as a real FPAA architecture.

As a solution to the first problem, there is a method for combining several responses, and optimizing the overall result quality while maintaining minimum satisfaction levels for each response: the desirability function [95, 96]. A desirability function is a mapping of the response to the range [0, 1], where 1 is the most desirable value and 0 is unacceptable. This mapping also helps with normalizing the responses that may have very different ranges of values. Each response can be transformed to [0, 1] range in a different way to reflect the characteristic of that response. For instance, one may want to achieve a target value for fc whereas the highest value possible is desired for any of the utilization responses. In this case, desirability function for fc can be defined such that it reaches 1 when fc value approaches the target fc, and goes down to 0 as fc goes farther away from fc in both directions. A maximum deviation from fc, say $\pm 10\%$ of the target value, can be set so that the corresponding desirability is 0 for fc values outside this range. Hence, the desirability

function for fc looks like a triangle with the apex at the target fc. For utilization, the desirability function may start as 0 at the minimum acceptable utilization value, and reach to 1 when the utilization is 100% (or any satisfactory value that going higher than this value does not matter anymore). In this case, desirability function will look like a ramp. It is also possible to shape the ramp by taking the powers of desirability. If the power is greater than 1, desirability function takes a concave shape, whereas a power in the range (0, 1) gives a convex shape to the desirability. If any value within a certain range is equally acceptable for a response, the desirability function can be set to look like a rectangle that rises to 1 at the lower bound of the range, and falls back to 0 at the higher bound. Figure 4.7 illustrates different desirability functions for the objective of maximization, minimizing deviation from a target value, and minimization. The effect of taking the powers of different regions is also demonstrated in Figure 4.7d.

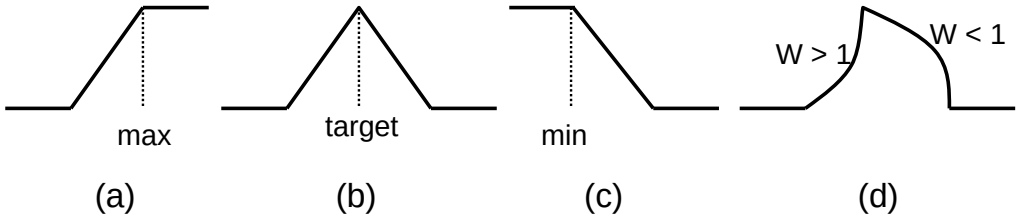


Figure 4.7. Desirability functions for different optimization goals: a) maximize, b) minimize deviation from a target value, c)minimize, d) minimize deviation from target, but with weights other than 1 on each side.

The major advantage of using desirability functions rather than the usual cost function based on the weighted sums of different costs is the ease of combining multiple objectives. When weighted sums are used, assigning weights to each response is not a straightforward process. The different numerical value ranges and importance of responses can be reflected to the weights to some degree; however, enforcing satisfactory results for all objectives may not always be possible. The unacceptable low values for a response may be compensated by the unacceptable high values of another when added up, erroneously giving an acceptable cost value. On the other hand, combining the desirability functions of different responses

is simply taking their geometric mean:

$$D = (\prod_{i}^{n} d_i)^{\frac{1}{n}} \tag{4.6}$$

Moreover, if each response i has a different level of importance, they can be assigned weights w_i as desirability powers:

$$D = (\prod_{i}^{n} d_i^{w_i})^{\frac{1}{\sum_i^n w_i}} \tag{4.7}$$

When the geometric mean of desirability functions are taken, any response that has an unacceptable value (i.e. desirability = 0) will make the overall desirability value 0; therefore, all responses must be satisfied simultaneously while finding the best combination of responses leading to the highest desirability.

By inspection of the various possible outcomes of the desirability transform depicted in Figure 4.7, it can be deduced that the desirability function for a response is not continuously derivable, leaving the traditional gradient-based optimizers unusable to solve this problem. Thus, the optimizers that can be used are limited to the derivative-free search algorithms. Pattern search (PS) [97, 98] and simulated annealing (SA) [68] algorithms fall into this category of optimizers and both are available in MATLAB; therefore, the polynomial models generated by the MBC toolbox, and the desirability functions that combine and reduce them into a single cost function can easily be linked to PS or SA optimizers to find the input vectors that result in the highest desirability values possible. Pattern search algorithm starts with an initial solution point, an initial pattern determined by the used search method, and a scalar parameter that controls the step length to iteratively find lower cost points, and move toward them until convergence, which is decided by the step length that is updated at each iteration. Simulated annealing also starts from an initial point, which is subject to random perturbation to find lower cost points nearby. In the initial steps of simulated annealing when the simulation temperature is assumed to be high, the moves that result in higher points may also be accepted with a probability that falls with decreasing temperature, preventing the optimizer from getting stuck at local minima. Both optimizers must iterate

though valid points at all times. It is also possible to have a combined optimization by using the result of one optimizer as the starting point of the other optimizer. In this study, the initial point was determined by taking the median value for each input factor; and it was observed that SA followed by PS resulted in the design points with highest desirability values compared to each optimizer alone, or PS followed by SA.

Another challenge is the necessity of defining the optimum architecture using discrete points; therefore, the optimization results must be constrained to having discrete values. If the optimizer does not support this constraint, one option may be first relaxing this constraint to find a solution, then rounding the continuous values in the solution vector to the nearest discrete values on either side of the value. For a vector with n dimensions, the number of vectors that can be found by rounding to both sides for each dimension can be as high as 2^n when no element of the vector takes a discrete value. Although this method may become infeasible as the number of input factors increase, since there are only 9 input factors in this study, evaluating the resulting 512 architectures in MATLAB was not prohibitively expensive. For the studies that require higher dimensionality, a sampling among these points can be considered. For instance, if the rounding error difference between rounding the continuous values to the nearest discrete values is larger than a threshold, rounding off to only one side may be chosen to reduce the number of possible combinations.

4.5.3 Results and Conclusions

To summarize, the objective function for the optimizer was derived by combining the desirability functions obtained from the polynomial regression models. The desirability functions were designed such that routability is maximized ensuring its value does not fall below 0.95 (rising ramp between 0.95 and 1), all utilization responses are maximized (rising ramps between minimum and maximum values of the available design data), `fc` and `av` equalized to the target metrics obtained by simulating ideal interconnect circuits (triangle with its apex at the target value), and `rp` is minimized (falling ramp between 0 and maximum value that can be tolerated for the circuit). The ideal metrics used to determine the

target values are presented in Table 4.9 along with the median values obtained from simulation of all architectures tested. The optimization results are then rounded to the nearest discrete values to obtain a valid input value. The vectors obtained as the result of rounding are all candidate FPAA architectures. Then, each candidate architecture is evaluated again, and sorted with respect to the desirability values. Since the polynomial regression models predict the responses with some error, it is safer to choose and verify several architectures rather than the best architecture alone. In this case, 20 best architectures are chosen to be tested again using GRASPER and SPICE. After evaluating the simulation results of these 20 architectures using desirability functions, the best architecture was chosen.

Table 4.9. Comparison of the filter metrics for circuits connected with ideal interconnects and the median of the routed circuit simulation results.

	blp8	c1lp7	c2lp5	elp4
ideal fc	9976	7924	8298	9527
median fc	9976	7924	8891	10690
ideal av	0.564	2.593	0.132	-0.035
median av	0.892	5.791	4.439	3.254
ideal rp	0.384	10.832	0.208	2.579
median rp	3.171	17.360	0.923	3.011

First, architectures were optimized with respect to each of the four circuits individually. This makes setting tight restrictions on each circuit's performance metrics possible. Tables 4.10, 4.11, 4.12, and 4.13 present the polynomial model and GRASPER-SPICE simulation results of the optimum architecture for the circuits blp8, c1lp7, c2lp5, and elp4 respectively. The individual desirability values for each circuit are represented by d, whereas D_{total} stands for the overall desirability when all circuits are considered. Of course, when only one circuit is considered for optimization, D_{total} has to be equal to d of only that circuit. When the architecture was optimized for each circuit individually, a maximum deviation of 0.67% from the target `fc` (53 Hz for c1lp7) was observed for the optimized filter based on the polynomial model. When evaluated with SPICE, the maximum deviation from target `fc` becomes 4.5% (374 Hz for c2lp5). `av` and `rp` values have less accuracy than `fc`; however,

from a design point of view they mostly remained within reasonable margins for the types of placed and routed circuits.

Table 4.10. Optimization of architectures with respect to blp8 only.

	blp8		c1lp7		c2lp5		elp4	
	model	sim	model	sim	model	sim	model	sim
fc	9978	9976	7136	7744	8383	7924	9754	9099
av	0.909	1.494	6.255	9.410	1.956	3.827	2.318	9.990
rp	2.003	3.073	9.300	10.754	0.994	1.250	2.651	5.430
swutil	1.099	1.299	1.093	1.196	1.103	1.262	0.992	1.092
wireutil	19.223	19.816	19.085	18.373	18.963	19.029	17.060	16.535
cmputil	13.132	13.281	13.135	13.281	14.699	14.844	13.150	13.281
d	0.740	0.728	0.000	0.600	0.657	0.567	0.626	0.000
D_tot	0.740	0.728	0.740	0.728	0.740	0.728	0.740	0.728

Table 4.11. Optimization of architectures with respect to c1lp7 only.

	blp8		c1lp7		c2lp5		elp4	
	model	sim	model	sim	model	sim	model	sim
fc	10487	10690	7871	7924	8339	7744	9507	11194
av	0.863	0.867	6.399	6.049	4.518	2.542	2.689	-0.393
rp	1.894	1.291	11.566	11.450	0.618	0.066	3.318	1.298
swutil	1.047	1.018	1.020	1.026	1.033	1.018	0.929	0.902
wireutil	16.133	15.572	15.813	15.693	15.659	15.328	14.166	13.625
cmputil	10.712	10.625	10.711	10.625	11.973	11.875	10.712	10.625
d	0.000	0.000	0.601	0.678	0.512	0.000	0.563	0.000
D_tot	0.601	0.678	0.601	0.678	0.601	0.678	0.601	0.678

Then, architectures were optimized for all circuits simultaneously. This was obviously harder than optimizing for each circuit individually, so the initial optimization efforts using the constraints from the individual optimizations for each circuit were unsuccessful. To achieve a solution, each circuit had to compromise to some extent, so that other circuits would also work within acceptable ranges. The maximum frequency deviation allowed was increased to 600 Hz for each circuit, and the results presented in Table 4.14 were obtained.

Table 4.12. Optimization of architectures with respect to c2lp5 only.

	blp8		c1lp7		c2lp5		elp4	
	model	sim	model	sim	model	sim	model	sim
fc	11041	10209	7564	7227	8334	7924	9092	8491
av	0.948	0.792	5.251	7.104	-0.477	0.478	1.496	0.113
rp	3.289	0.613	12.176	9.047	0.487	1.543	2.390	4.313
swutil	1.075	1.093	1.071	1.075	1.070	1.039	0.989	0.994
wireutil	15.943	16.145	15.857	15.904	15.619	15.181	14.314	14.458
cmputil	13.147	13.281	13.147	13.281	14.706	14.844	13.157	13.281
d	0.000	0.000	0.000	0.000	0.619	0.439	0.000	0.000
D_tot	0.619	0.439	0.619	0.439	0.619	0.439	0.619	0.439

Table 4.13. Optimization of architectures with respect to elp4 only.

	blp8		c1lp7		c2lp5		elp4	
	model	sim	model	sim	model	sim	model	sim
fc	11648	11454	8288	8491	8634	9310	9518	9527
av	1.089	2.007	6.130	7.299	1.361	2.744	1.787	2.439
rp	3.406	5.082	15.529	13.765	1.492	0.442	3.394	4.173
swutil	1.517	1.649	1.462	1.575	1.479	1.438	1.347	1.364
wireutil	19.010	19.891	18.444	19.074	18.275	17.302	16.658	16.349
cmputil	13.138	13.281	13.145	13.281	14.706	14.844	13.157	13.281
d	0.000	0.000	0.000	0.000	0.000	0.000	0.690	0.677
D_tot	0.690	0.677	0.690	0.677	0.690	0.677	0.690	0.677

As Table 4.14 demonstrates, when the architecture is optimized for all circuits simultaneously, `fc` deviates from the target `fc` between 1.6% and 5.6% based on the polynomial models of the circuits, and between 2.3% and 6.7% when evaluated using SPICE. Note that D_{total} in this case is the geometric mean of the individual d for each circuit.

Table 4.14. Optimization of architectures with respect to all circuits.

	blp8		c1lp7		c2lp5		elp4	
	model	sim	model	sim	model	sim	model	sim
fc	10531	10446	7400	7396	8016	7924	9379	9749
av	1.286	1.959	5.057	6.234	1.242	3.939	2.185	-0.166
rp	3.573	3.490	11.129	11.908	0.214	0.730	2.222	6.559
swutil	1.096	1.147	1.056	1.056	1.039	1.065	0.962	0.965
wireutil	18.608	19.468	18.168	18.067	17.482	17.927	16.037	16.246
cmputil	13.136	13.281	13.145	13.281	14.705	14.844	13.155	13.281
d	0.368	0.551	0.422	0.538	0.586	0.546	0.496	0.583
D_tot	0.461	0.554	0.461	0.554	0.461	0.554	0.461	0.554

Table 4.15 presents the 9 input factors that define the resulting architectures obtained in different optimizations. D_{model} and D_{sim} represent the overall desirabilities of the architectures evaluated by the polynomial regression model and SPICE simulations respectively. Since a probabilistic algorithm (SA) was used for optimization, different architectures may be obtained with different attempts, each working within the limits that can be controlled using the desirability functions.

Table 4.15. Architectures found as results of different optimizations.

	sw	hg	v8	v4	v2	v1	hn	OTA	cap	D_{model}	D_{sim}
b only	0.875	8	0	2	11	4	1	1	1	0.740	0.728
c1 only	0.875	3	5	6	4	7	1	1	2	0.601	0.678
c2 only	0.750	3	9	4	8	5	4	1	1	0.619	0.439
e only	0.500	5	11	5	7	3	2	1	1	0.690	0.677
all	0.750	7	12	1	5	3	3	1	1	0.461	0.554

The objective of this study is to develop and present a systematic way for exploration and evaluation of the optimum architecture for a given set of applications. The architecture

demands vary by the circuit classes and sizes, so their inclusion in the study will result in different architectures to accommodate all circuits. Yet, the method presented here is easily extendable to different circuit classes, as long as the performance of those circuits can be efficiently evaluated within an automated platform as well.

CHAPTER 5

CONCLUSION AND FUTURE DIRECTIONS

In this study, a placement and routing platform was developed for mapping analog circuits onto a target FPAA device. The efforts to develop this platform started as the RASPER tool which supported only RASP2.7 and RASP2.8 architectures, and eventually evolved into the GRASPER tool, which supports a wide range of architectures including, but not limited to RASP2.8 and RASP2.9. Replacement of manual switch list generating efforts with this tool makes the FPAA technology more accessible to the design community by reducing the design time as well as increasing the reliability of the design process. The tool has been used in Georgia Tech research groups both for testing new FPAA architectures and prototyping circuits on these architectures. The tool has also been used in the class laboratories and workshops, establishing a community that grows steadily.

GRASPER allows input circuit netlist and user constraints description in a SPICE-compatible format. The tool is modular, and can easily be integrated into systems that perform other phases of the physical design. GRASPER is also based on a topology independent routing resource graph, which makes it very flexible with a hardware description format that was developed to describe a wide range of architectures.

Using floating-gate transistors as switches brings new opportunities to the circuit design by utilizing the routing fabric as functional units rather than just for connection. Unlike the binary operation of other switch technologies, floating-gate switches can be set to operate in the intermediate regions between the extreme on and off; therefore, they can be used as circuit components rather than just routing elements. Some circuits using this feature of the switch elements, such as the diffusor circuits, were introduced in Chapter 2. On the other hand, such circuits bring new challenges that were not previously addressed in physical design. In the case of diffusors, GRASPER encounters floating nets (i.e. nets with no source and sink wires described at the beginning of routing) since the pins of all nets

are not determined at the end of placement. These problems were addressed and solved in Chapter 2.

GRASPER includes a module that can extract the routing parasitics of the mapped circuits, allowing simulation of the routed circuits using third party simulator software. This can be useful for troubleshooting in case the routed circuits do not perform as expected. It is not possible to probe every net of a circuit programmed on the FPAA, but this is not a problem in simulation. Simulation results can also be used to guide the performance optimization by trimming the configuration elements on the chip as well. Simulation models are not hardcoded into the tool; therefore, models of different accuracy and complexity levels can be used for different purposes. Using a technology file, new CAB component models and process parameters for the transistors can be entered for different FPAAs. The models that were available during the course of this study can be replaced with better models in the future, allowing more reliable simulation results.

The model order of the parasitic extracted circuit is left to the user control with the option of choosing the minimum time constant, which is inversely proportional to the maximum frequency, of interest. By setting this value at the right level for the circuit's range of operation, the complexity of the resulting circuit is not increased unnecessarily.

GRASPER optimizes the analog circuit performance by taking the routing parasitics into account, and minimizing the capacitance deviation from the intended values for each net. Although the target net capacitance would be 0 in most cases, some circuits may require certain capacitance values at each net for correct operation. Thus, the capacitance coming from the routing effects may be used to advantage, reducing the need for drawn CAB capacitors.

Parasitic modeling and flexible architecture description in GRASPER can be very useful in the design of new FPAA architectures as well. Different topologies with varying resources can be tested to determine the efficiency and the effectiveness of architectures. For this reason, it is beneficial to prepare the device files for the planned architectures and test

them with the target circuits before designing the layout and fabricating the architectures, significantly saving resources. The architectures can also be optimized for various circuit classes and sizes using the systematic architecture exploration and evaluation method as shown in Chapter 4.

5.1 Directions for Future Research

This research can be continued to explore and improve the following items:

- An integrated fast simulation engine can be developed and used in guiding the performance optimization.
- Power analysis and optimization for placement and routing of circuits on FPAA can be done.
- Different phases of physical design can be implemented, so that analog systems can be designed on the FPAA starting from the level description of circuit behavior and specifications. These steps can be analog synthesis as the analog domain equivalent of logic synthesis in FPGA synthesis, and technology mapping to design the circuit using the available FPAA components with the non-ideal effects of the CAB components and routing resources taken into account.
- Optimization tasks can be distributed between the different levels of physical design. These tasks may involve optimum I/O pin ordering and selection, and optimization of component configuration via bias currents considering the possible routing parasitic effects.
- A circuit library may be developed to make the design of larger systems easier.
- Benchmark suites should be developed for stable code improvement.
- Full digital support with inclusion of directional switches and digital performance

metrics extraction can be added, so that FPGAs can also be developed and evaluated, eventually leading to FPMA support.

- With inclusion of digital circuit support, there are opportunities in optimization for asynchronous circuit design by intelligently controlling the delays on floating-gate based switches.

- FPAA architecture design can be automated from the GRASPER device files. It's a good practice to test the conceptual architectures before proceeding with their design and fabrication. Having such an automation tool will make the design process more efficient and reliable.

APPENDIX A

GRASPER USER MANUAL

A.1 Running GRASPER

GRASPER can be executed by either command line entry or clicking on the executable file. If input circuit file is not provided at execution, user will be prompted to enter one:

> **grasper** <path_to_input_file>

A.2 Input files

GRASPER requires three files to place and route a circuit onto the target FPAA. These files are:

- A device file that contains FPAA architecture description. This file is described in further detail in Appendix B.
- A technology file that contains SPICE model information for components in the FPAA. Each FPAA component may be a single electrical element like a transistor, or a circuit block like an OTA. All used FPAA components must be specified as SPICE subcircuits in this file.
- An input file that contains description of the circuit that will be mapped onto the FPAA. This file uses the same circuit description format as SPICE.

GRASPER input file is written in such a way that it can also be simulated by SPICE software as well. However, all element types that can be used in a SPICE file are not available in GRASPER. User may enter only the components available in the device file for a given FPAA architecture as subcircuit instantiations similar to SPICE. Subcircuit models for these components must be described in the technology file as mentioned above. Parameters can be used to configure the FPAA components if needed.

Example: X1 innet_p innet_n outnet OTA PARAMS: Ib=1u

X2 pin_d pin_g pin_s NFET_RASP2_8

Xcap1 p1 p2 C500F

GRASPER allows user-defined subcircuits that are built using the FPAA components to describe larger circuit blocks. It is also possible to include circuits described in a separate file using the SPICE command:

.include <include_file>

so that design libraries can be easily maintained and used.

Like in SPICE, any portion of a line after "*****" is commented out unless that line starts with the sequence "∗ >>" to enter GRASPER specific commands, which is explained in the next section.

A.3 GRASPER commands

GRASPER commands are the keywords incompatible with SPICE, and they are entered following the keyword "∗ >>". Below is a list of various GRASPER commands:

∗ >> **devicefile** <filename>

Specifies the path to device file.

∗ >> **project** <directoryname>

Specifies the path to project directory. All output and log files are saved in here.

∗ >> **pin** <pin_info> net <net_name>

Specifies pin assignments for each I/O net in the circuit. If chip name is not specified in the pin_info field, it is assumed that the first chip defined in the device file will be used.

Example: ∗ >> pin io_lt 0 net nInp

∗ >> pin chip0 io_lt 1 net nInn

∗ >> pin chip1 io_rt 0 net nOut

∗ >> **place** <circuit_component_name> into <FPAA_component_name>

Specifies FPAA component assignments for each circuit component. FPAA component

names are a combination of chip_name, cab_name, and component index within its cab.

Example: * >> place xotap into chip0 cab3_0 0

* >> place xotap into chip1 cab4_1 0

* >> **route** <net_name> into <cab_name>

Specifies switch assignments for each I/O net in the circuit. If cab_name field does not include name of the chip it belongs to, it is assumed that the first chip defined in the device file will be used.

Example: * >> route net net_in1 0 14 0 52 14 14

* >> route swe Xswe1 232 46

* >> **option** <option_name> <option_value>

Specifies options that user may choose to carry out alternate operations when running GRASPER. Option command is explained in more detail in the next section.

A.4 Options

To customize the GRASPER results, user may specify various options by the following GRASPER command:

* >> **option** <option_name> <option_value>

When using this command, option_name is a required field while option_value is optional; if only option_name is specified, option_value is assumed to be 1. Options that are unspecified use default values.

The following options are used to display a group of elements or information related to GRASPER operation:

- **displaycmp** Lists FPAA components.
- **displaynets** Lists nets in the circuit graph.
- **displaypins** Lists pins in the circuit graph.
- **displaycells** Lists cells in the circuit graph.

- **displayiocells** Lists cells corresponding to the I/O ports.
- **displayglobalcells** Lists cells corresponding to the global nets.
- **displayplacementstats** Lists number of FPAA components and CABs used.
- **displayswnets** Lists routing switches and switch elements (swe) with associated nets; configuration switches with associated cells.
- **displayall** Lists all circuit elements and FPAA components.
- **displayswitchcount** Displays number of wires and switches used.
- **displaytiming** Displays how long each phase takes to complete.

These options control how and where to log placement and routing information:

- **log2file** Logs all information in the <circuit_name>.log file under the project directory if 1; to the terminal output otherwise (default 1).
- **logdevice** Logs FPAA related information after device setup.
- **logsetup** Logs circuit related information after circuit graph setup.
- **logplacement** Logs how circuit and device elements are affected by placement.
- **logrouting** Logs how circuit and device elements are affected by routing.
- **logall** Logs all phases.

These options determine additional (optional) output files:

- **routingfile** Writes the routing results in a netlist file where each net is assigned a routing constraint denoting the switches it was routed through previously.
- **placementfile** Writes the placement results in a netlist file where each circuit component is assigned a placement constraint denoting the FPAA component it was placed to.

- **placeonly** Writes a placement file and exits before routing.
- **extractedfile** Writes a parasitic extracted SPICE file for simulation of routing effects.

These options are used to choose between the alternate methods for placement, routing, and parasitic extraction:

- **cabrankorder** Chooses one of the four schemes for visiting FPAA components for each circuit component. Circuit components tend to cluster close to the initial components visited. 0 (default): forward (i.e. from bottom-left CAB to top-right CAB), 1: backward, 2: forward/backward alternating for each circuit component, 3: random.
- **reducedorder** simplifies the parasitic extracted circuit (default 1).
- **minResistance** minimum interconnect resistance in parasitic extraction. Any resistance below this value is assumed to be 0 (default 1e-6).
- **minTau** minimum period of the signals that can be reliably simulated after model order reduction is applied to parasitic extracted circuits (default 100e-9).

A.5 Output files

The following files are generated when GRASPER runs. They can be found in the project directory:

- <**circuitname**>**.out** Coordinates of the switches that establish the circuit connection can be found in this file.
- <**circuitname**>**.log** Information collected during all phases of file parsing, placement, and routing can be found in this file. User may decide what information to be reported via the options explain in the previous section. If placement and routing cannot be completed, error messages about the reason for the reason of failure and a

partial routing solution with the list of switches used for routing until that point can also be found in this file. Including displayall and logall options is recommended for a detailed inspection of the problem in such a case.

- <**circuitname**>_**placed.sp** Placement results are saved in this file, which can also be used as an input file for GRASPER. FPAA components associated with the circuit components are specified using the placement constraint (∗ >> place), so placement step is skipped when this file is used as GRASPER input.

- <**circuitname**>_**routed.sp** Routing results are saved in this file. In addition to the placement results, switches associated with each net or swe are specified using the routing constraint (∗ >> route).

- <**circuitname**>_**ext.sp** This file is a SPICE netlist that contains extracted parasitic interconnects, and can be used to simulate the routing effects ono the performance of the circuit. Accuracy of simulation results rely on the models provided in the technology file.

A.6 Generating SPICE netlists from switch file

Circuit connectivity of the original SPICE netlist can be recreated using the -s option of GRASPER from the command line. Since the original component and netnames cannot be retrieved from the output switch file, the resulting netlist will not be identical to the original netlist lexically. On the other hand, circuit connectivity remains the same. Device file information is also not embedded in the switch file; therefore, it must be provided at the command line entry. The resulting SPICE netlist is displayed at the standard output. Note that this option is available only from the command line.

Format: > grasper -s switchfile -d devicefile

APPENDIX B

GRASPER ARCHITECTURE DESCRIPTION FORMAT

GRASPER requires a device file that defines the FPAA architecture by describing the interconnect topology, component resources, and routing resources. Using this file, a wide range of FPAA architectures can be used for placement and routing of circuits.

In the device file, vertical and horizontal dimensions are divided into grids that represent the minimum distance between any two switches in that orientation. These grid numbers are used to describe the physical locations on the FPAA. Coordinates (r,c) of a point are given as the vertical grid number (row) first, and the horizontal grid number (column) next. It is possible to enter vectors of numbers by providing the initial number, the final number, and an optional increment (default increment value is 1 if not provided). For instance ”0:8:2” represents the vector [0 2 4 6 8]. To describe a vector of points, either row number or column number (both not both at the same time) can be vectorized as well.

There are three keyword types in the FPAA device file:

- **element types** that declare the properties of FPAA resource types.
- **elements** that instantiate FPAA resources.
- **other commands** used for describing the geometry or other aspects of certain FPAA resources.

B.1 element types

The following keywords are used to describe the element types.

iopintype name { cost pin(0) cost_value; };

Descriptions of chip I/O pins. Cost for connecting to each pin can be entered via the cost keyword. Example:

iopintype IOPAD { cost pin(0) 1000; };

iopingroup name orientation grid_number

I/O pins that are horizontally or vertically aligned whith each other are grouped using this command. Having this command makes it easier for humans to interpret I/O pin locations. Use h for horizontal and v for vertical orientations, followed by the horizontal or vertical coordinates (grid numbers) for the pins in group. Examples:

iopingroup io_lt h 0

iopingroup io_up v 366

cmptype name number_of_pins { param ... ; cost pin(x)... ; };

Names and pin numbers for CAB component types are declared using this command. Pin cost can be entered using the cost keyword for each pin (in the same fashion as iopintype). If component has parameters, enter them in the same order as the circuit description in SPICE using param keyword (see below for more details on using param). Example:

cmptype FGOTA 3 {

param Ib;

param fgv2i(valn);

param fgv2i(valp);

};

swtype name { ... };

Type info for switches used in FPAA. Using the format keyword, how switches will be written in the output file can be shaped (see below for more details on using format). Using the cap keyword, off-capacitance for switch can be entered. Example:

swtype RSW { format r c const(1.8); cap 1e-15; };

swetype name { param ...; format ...; }

Type info for switch elements (SWE) supported in FPAA. Since this element type is a physical switch, format keyword can be used similar to its use for swtype. Since SWE correspond to components in the circuit description, param keyword can be used similar to its use for cmptype. Example:

```
swetype FGE1 {
param vg;
format r c val(0);
};
```

wiretype name orientation length { res ...; cap ...; };

Type info for wires of different length and orientation in FPAA. Length of wire is specified in number of spanned grids. Resistance and capacitance of wire segments per grid can also be specified using res and cap keywords. Example:

```
wiretype hgwire hor 142 { res 0.5; cap 1e-17; };
```

cabtype name number_of_pins vertical_length horizontal_length { ... };

Type info for different CABs that can be found in FPAA. Components of the CABs are instantiated within the braces of cabtype. Example:

```
cabtype cab1a 41 41 2  ... ;
```

chiptype name vertical_length horizontal_length { ... }

Type info for chip(s) that can be found in FPAA (or multi-chip board). CABs, wires, switches, and I/O pins of the chip(s) are instantiated within the braces of chiptype. Example:

```
chiptype ct1 367 144 {
cab ...;
wire ...;
```

switch ...;

iopin ...;

};

B.2 elements

The following keywords are used to describe the elements.

cmp cmptype cabpinnumbers configuration_switches;

Instantiates components in a cabtype. Configuration switches can be added in the order of parameters they are associated with. Each configuration switch is introduced by the switch type name and switch coordinates relative to the CAB that contains the component. Example:

cmp OTA 39 40 38 CSW(38,0);

wire wiretype(id) bottom_left_point;

Instantiates wires in FPAA by providing the wire type, an id for the wire within the context of its type, and the coordinates of bottom left point of the wire. Example:

wire hgwire(0:3) (0:3,0);

cab cabname cabtype bottom_left_point { pins(...) wires; };

Instantiates CABs in a chip. Each pin of the CAB must be connected to a wire. Example:

cab cab0_0 cab1a (4,32) { pins(0:40) clwire1(0:40); }; **switch switch_type at matrix ...;**

or

switch switch_type wire1_id to wire2_id at ...;

Instantiates routing switches in a chip. Entry can either be a matrix that marks the switches to be placed if two perpendicular wires intersect at that point, or between any two wires by indicating the switch coordinates. Example:

switch RSW at matrix m3 (4:326:46,0);

switch RSW vlwire(4:7) to vlwire(0:3) at (45,24:27);

iopin iopintype iopingroup_id wire_id;

Instantiates I/O pins of a chip. Each I/O pin is numbered from an iopingroup. Each I/O pin is connected to a wire in the same chip. I/O pin - wire associations can be vectorized. Example:

iopin IOPAD io_lt(2:8) hgwire(4:28:4);

chip chipname chiptype bottom_row left_column;

Instantiates a chip of a chiptype also described in the device file. Example:

chip chip0 ct1 0 0;

B.3 other commands

matrix name row_number column_number { ... };

Defines a matrix to describe multiple points easily. Points are described for each row. Vectorization is useful when there are repeating patterns. Example:

matrix m1 4 32 {

row(0:2:2) 0:30:2;

row(1:3:2) 1:31:2;

};

merge wire1 wire2;

Establishes physical connection between two wires. Example:

merge hnwire(0:31) hnwire(32:63);

global netname wire

Declares global net names associated with wire(s). This command is useful for power connections. If multiple wires are associated with the same net using global command, all of them are merged. Example:

global Vdd vgwire(1:37:12);

format ...;

Describes the format of the switches to be written in the output file using the keywords r for row, c for column, const for a constant value, and val for a variable value to be received from the parameter list of the associated component (for configuration switches or switch elements). Example:

format r c const(1.8);

format r c val(0);

param ...;

Allows parameter entry for CAB components or switch elements. Parameter values can be determined either directly from the circuit netlist line, or by transforming the netlist line value using the built in fgv2i function that finds current equivalent of a voltage. Example:

param Ib;

param fgv2i(valn);

APPENDIX C

SPICE NETLISTS FOR SOME OF THE GMC FILTERS USED AS TEST CIRCUITS

Netlist for blp8:

*gmC filter netlist for butterworth low 10000 Hz (order 8)

vin 2 0 dc 1.2 ac 1

vref 1 0 1.2

C1 3 0 1p

X1 2 1 3 OTA PARAMS: Ib=4.713496e-009

X3 1 3 3 OTA PARAMS: Ib=9.303887e-009

X5 1 4 3 OTA PARAMS: Ib=4.713496e-009

C4 4 0 1p

X7 3 1 4 OTA PARAMS: Ib=4.713496e-009

C5 5 0 1p

X9 4 1 5 OTA PARAMS: Ib=4.713496e-009

X11 1 5 5 OTA PARAMS: Ib=7.872838e-009

X13 1 6 5 OTA PARAMS: Ib=4.713496e-009

C6 6 0 1p

X15 5 1 6 OTA PARAMS: Ib=4.713496e-009

C7 7 0 1p

X17 6 1 7 OTA PARAMS: Ib=4.713496e-009

X19 1 7 7 OTA PARAMS: Ib=5.241477e-009

X21 1 8 7 OTA PARAMS: Ib=4.713496e-009

C8 8 0 1p

X23 7 1 8 OTA PARAMS: Ib=4.713496e-009

C9 9 0 1p

X25 8 1 9 OTA PARAMS: Ib=4.713496e-009

X27 1 9 9 OTA PARAMS: Ib=1.828414e-009

X29 1 10 9 OTA PARAMS: Ib=4.713496e-009

C10 10 0 1p

X31 9 1 10 OTA PARAMS: Ib=4.713496e-009

*gmC filter with 8 caps and 32 OTA

.include fpaa_tech.sp

Xout 10 filter_output filter_output OTA PARAMS: Ib=10u

.control

op

let isrc = @Vddsrc[i]

ac dec 100 500 500k

*plot vdb(filter_output)

let outnode = v(filter_output)

write rawfile.txt isrc outnode

.endc

*configuration files (can also be entered in the command line)

* >> devicefile archgen.dev

* >> project work

*i/o connections

* >> pin io_lt 0 net 1

* >> pin io_lt 1 net 2

* >> pin io_rt 1 net filter_output

*user constraints

* >> option displayswitchcount

* >> option displaytiming

* >> option displayplacementstats

* >> option displaycells

* >> option displaynets

* >> option displayswnets

* >> option logall

* >> option extractedfile

.end

Netlist for c1lp7:

*gmC filter netlist for chebyshev low 10000 Hz (order 7)

vin 2 0 dc 1.2 ac 1

vref 1 0 1.2

C3 3 0 1p

X1 2 1 3 OTA PARAMS: Ib=4.713496e-009

X3 1 3 3 OTA PARAMS: Ib=1.197053e-009

C4 4 0 1p

X5 3 1 4 OTA PARAMS: Ib=4.713496e-009

X7 1 4 4 OTA PARAMS: Ib=2.165271e-009

X9 1 5 4 OTA PARAMS: Ib=2.364818e-009

C5 5 0 1p

X11 4 1 5 OTA PARAMS: Ib=2.364818e-009

C6 6 0 1p

X13 5 1 6 OTA PARAMS: Ib=9.414782e-009

X15 1 6 6 OTA PARAMS: Ib=1.495130e-009

X17 1 7 6 OTA PARAMS: Ib=3.872610e-009

C7 7 0 1p

X19 6 1 7 OTA PARAMS: Ib=3.872610e-009

C8 8 0 1p

X21 7 1 8 OTA PARAMS: Ib=5.737742e-009

X23 1 8 8 OTA PARAMS: Ib=5.273019e-010

X25 1 9 8 OTA PARAMS: Ib=4.751561e-009

C9 9 0 1p

X27 8 1 9 OTA PARAMS: Ib=4.751561e-009

X29 1 10 10 OTA PARAMS: Ib=2.000000e-005

X31 9 1 10 OTA PARAMS: Ib=1.976841e-007

*gmC filter with 7 caps and 32 OTAs

.include fpaa_tech.sp

Xout 10 filter_output filter_output OTA PARAMS: Ib=10u

.control

op

let isrc = @Vddsrc[i]

ac dec 100 500 500k

*plot vdb(filter_output)

let outnode = v(filter_output)

write rawfile.txt isrc outnode

.endc

*configuration files (can also be entered in the command line)

* >> devicefile archgen.dev

* >> project work

*i/o connections

* >> pin io_lt 0 net 1

* >> pin io_lt 1 net 2

* >> pin io_rt 1 net filter_output

*user constraints

* >> option displayswitchcount

* >> option displaytiming

* >> option displayplacementstats

* >> option displaycells

* >> option displaynets

* >> option displayswnets

* >> option logall

* >> option extractedfile

.end

Netlist for c2lp5:

*gmC filter netlist for inverse chebyshev low 10000 Hz (order 5)

vin 2 0 dc 1.2 ac 1

vref 1 0 1.2

C3 3 0 1p

X1 2 1 3 OTA PARAMS: Ib=4.713496e-009

X3 1 3 3 OTA PARAMS: Ib=7.451028e-009

C4 4 0 1p

X5 3 1 4 OTA PARAMS: Ib=4.713496e-009

X7 1 4 4 OTA PARAMS: Ib=6.484821e-009

X9 1 5 4 OTA PARAMS: Ib=5.453171e-009

C5 5 0 1p

X11 4 1 5 OTA PARAMS: Ib=5.453171e-009

C6 6 0 1p

X13 3 1 6 OTA PARAMS: Ib=4.713496e-009

X15 1 4 6 OTA PARAMS: Ib=6.484821e-009

X17 5 1 6 OTA PARAMS: Ib=6.373639e-009

X19 1 6 6 OTA PARAMS: Ib=1.404063e-009

X21 1 7 6 OTA PARAMS: Ib=4.117681e-009

C7 7 0 1p

X23 6 1 7 OTA PARAMS: Ib=4.117681e-009

X25 1 8 8 OTA PARAMS: Ib=2.000000e-005

X27 3 1 8 OTA PARAMS: Ib=5.868504e-006

X29 1 4 8 OTA PARAMS: Ib=1.013566e-005

X31 5 1 8 OTA PARAMS: Ib=9.834229e-006

X33 1 6 8 OTA PARAMS: Ib=9.223396e-007

X35 7 1 8 OTA PARAMS: Ib=1.336546e-006

*gmC filter with 5 caps and 36 OTAs

.include fpaa_tech.sp

Xout 8 filter_output filter_output OTA PARAMS: Ib=10u

.control

op

let isrc = @Vddsrc[i]

ac dec 100 500 500k

*plot vdb(filter_output)

let outnode = v(filter_output)

write rawfile.txt isrc outnode

.endc

*configuration files (can also be entered in the command line)

* >> devicefile archgen.dev

* >> project work

*i/o connections

* >> pin io_lt 0 net 1

* >> pin io_lt 1 net 2

* >> pin io_rt 1 net filter_output

*user constraints

* >> option displayswitchcount

* >> option displaytiming

* >> option displayplacementstats

* >> option displaycells

* >> option displaynets

* >> option displayswnets

* >> option logall

* >> option extractedfile

.end

Netlist for elp4:

*gmC filter netlist for elliptic low 10000 Hz (order 4)

vin 2 0 dc 1.2 ac 1

vref 1 0 1.2

C3 3 0 1p

X1 2 1 3 OTA PARAMS: Ib=4.713496e-009

X3 1 3 3 OTA PARAMS: Ib=4.665900e-009

X5 1 4 3 OTA PARAMS: Ib=3.911073e-009

C4 4 0 1p

X7 3 1 4 OTA PARAMS: Ib=3.911073e-009

C5 5 0 1p

X9 2 1 5 OTA PARAMS: Ib=4.713496e-009

X11 1 3 5 OTA PARAMS: Ib=4.665900e-009

X13 4 1 5 OTA PARAMS: Ib=2.480647e-008

X15 1 5 5 OTA PARAMS: Ib=6.740603e-010

X17 1 6 5 OTA PARAMS: Ib=4.828512e-009

C6 6 0 1p

X19 5 1 6 OTA PARAMS: Ib=4.828512e-009

X21 1 7 7 OTA PARAMS: Ib=2.000000e-005

X23 1 3 7 OTA PARAMS: Ib=5.252638e-007

X25 4 1 7 OTA PARAMS: Ib=6.089450e-006

X27 1 5 7 OTA PARAMS: Ib=5.780755e-008

X29 6 1 7 OTA PARAMS: Ib=1.373561e-007

X31 2 1 7 OTA PARAMS: Ib=5.321490e-007

*gmC filter with 4 caps and 32 OTAs

.include fpaa_tech.sp

Xout 7 filter_output filter_output OTA PARAMS: Ib=10u

.control

op

let isrc = @Vddsrc[i]

```
ac dec 100 500 500k
*plot vdb(filter_output)
let outnode = v(filter_output)
write rawfile.txt isrc outnode
.endc

*configuration files (can also be entered in the command line)
* >> devicefile archgen.dev
* >> project work

*i/o connections
* >> pin io_lt 0 net 1
* >> pin io_lt 1 net 2
* >> pin io_rt 1 net filter_output

*user constraints
* >> option displayswitchcount
* >> option displaytiming
* >> option displayplacementstats
* >> option displaycells
* >> option displaynets
* >> option displayswnets
* >> option logall
* >> option extractedfile

.end
```

REFERENCES

[1] H. Chen, I. Li-Da Huang, M. Liu, and M. Wong, "Simultaneous Power Supply Planning and Noise Avoidance in Floorplan Design," *IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems*, vol. 24, no. 4, 2005.

[2] T. Hall, C. Twigg, P. Hasler, and D. Anderson, "Developing large-scale field-programmable analog arrays," *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, 2004.

[3] P. Hasler, C. Diorio, B. Minch, and C. Mead, "Single Transistor Learning Synapses," *Advances In Neural Information Processing Systems*, pp. 817–826, 1995.

[4] M. Kucic, P. Hasler, J. Dugger, and D. Anderson, "Programmable and adaptive analog filters using arrays of floating-gate circuits," in *2001 Conference on Advanced Research in VLSI* (E. Brunvand and C. Myers, eds.), pp. 148–162, IEEE Computer Society, March 2001.

[5] M. Kucic, A. Low, P. Hasler, and J. Neff, "A programmable continuous-time floating-gate Fourier processor," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on [see also Circuits and Systems II: Express Briefs, IEEE Transactions on]*, vol. 48, no. 1, pp. 90–99, 2001.

[6] J. Gray, C. Twigg, D. Abramson, and P. Hasler, "Characteristics and programming of floating-gate pFET switches in an FPAA crossbar network," *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pp. 468–471, 2005.

[7] A. Arbel, "Current operated nucleonic modules," *Nuclear Instruments and Methods*, vol. 32, pp. 341–346, 1965.

[8] R. Joetten, T. Web, J. Wolters, H. Ring, and B. Bjoernsson, "A New Real Time Simulator for Power System Studies," *IEEE Transactions on Power Apparatus and Systems*, pp. 2604–2611, 1985.

[9] D. Vallancourt and Y. Tsividis, ""Timing-Controlled Switched Analog Filters with Full Digital Programmability."," *IEEE ISCAS 1987*, pp. 329–333, 1987.

[10] S. Eberhardt, T. Duong, and A. Thakoor, "Design of parallel hardware neural network systems from customanalog VLSIbuilding block'chips," in *Neural Networks, 1989. IJCNN., International Joint Conference on*, pp. 183–190, 1989.

[11] W. Fisher, R. Fujimoto, and R. Smithson, "A programmable analog neural network processor," *IEEE Transactions on Neural Networks*, vol. 2, no. 2, pp. 222–229, 1991.

[12] J. Van der Spiegel, P. Mueller, D. Blackman, P. Chance, C. Donham, R. Etienne-Cummings, and P. Kinget, "An analog neural computer with modular architecture for real-time dynamic computations," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 1, pp. 82–92, 1992.

[13] J. Choi, S. Bang, and B. Sheu, "A programmable analog VLSI neural network processor for communication receivers," *IEEE Transactions on Neural Networks*, vol. 4, no. 3, pp. 484–495, 1993.

[14] R. Chang, B. Sheu, J. Choi, and D. Chen, "Programmable-weight building blocks for analog VLSI neural network processors," *Analog Integrated Circuits and Signal Processing*, vol. 9, no. 3, pp. 215–230, 1996.

[15] L. Chua and L. Yang, "Cellular neural networks: Theory," *IEEE Transactions on circuits and systems*, vol. 35, no. 10, pp. 1257–1272, 1988.

[16] T. Roska and L. Chua, "The CNN universal machine: an analogic array computer," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 40, no. 3, pp. 163–173, 1993.

[17] M. Sivilotti, "A dynamically configurable architecture for prototyping analog circuits," in *Proceedings of the fifth MIT conference on Advanced research in VLSI table of contents*, pp. 237–258, MIT Press Cambridge, MA, USA, 1988.

[18] F. Kub, K. Moon, I. Mack, and F. Long, "Programmable analog vector-matrix multipliers," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 1, pp. 207–214, 1990.

[19] E. Lee and P. Gulak, "Prototype Design of a Field Programmable Analog Array," *Aug*, vol. 30, pp. 1–2, 1990.

[20] E. Lee and P. Gulak, "A CMOS field-programmable analog array," *Solid-State Circuits, IEEE Journal of*, vol. 26, no. 12, pp. 1860–1867, 1991.

[21] E. Lee and P. Gulak, "Field programmable analogue array based on MOSFET transconductors," *Electronics Letters*, vol. 28, no. 1, pp. 28–29, 1992.

[22] E. Lee and P. Gulak, "A transconductor-based field-programmable analog array," in *Solid-State Circuits Conference, 1995. Digest of Technical Papers. 42nd ISSCC, 1995 IEEE International*, pp. 198–199, 1995.

[23] Z. Czarnul, "Novel MOS resistive circuit for synthesis of fully integrated continuous-time filters," *IEEE Transactions on Circuits and Systems*, vol. 33, no. 7, pp. 718–721, 1986.

[24] E. Pierzchala, M. Perkowski, P. Van Halen, and R. Schaumann, "Current-mode amplifier/integrator for a field-programmable analogarray," *Solid-State Circuits Conference, 1995. Digest of Technical Papers. 42nd ISSCC, 1995 IEEE International*, pp. 196–197, 1995.

[25] E. Pierzchala, M. Perkowski, and S. Grygiel, "A field programmable analog array for continuous, fuzzy, andmulti-valued logic applications," in *Multiple-Valued Logic, 1994. Proceedings., Twenty-Fourth International Symposium on*, pp. 148–155, 1994.

[26] C. Premont, R. Grisel, N. Abouchi, and J. Chante, "A Current Conveyor based Field Programmable Analog Array," *Analog Integrated Circuits and Signal Processing*, vol. 17, no. 1, pp. 105–124, 1998.

[27] X. Quan, S. Embabi, and E. Sanchez-Sinencio, "A current-mode based field programmable analog array architecture for signal processing applications," in *Custom Integrated Circuits Conference, 1998., Proceedings of the IEEE 1998*, pp. 277–280, 1998.

[28] A. Bratt and I. Macbeth, "Design and implementation of a field programmable analogue array," in *Proceedings of the 1996 ACM fourth international symposium on Field-programmable gate arrays*, pp. 88–93, ACM New York, NY, USA, 1996.

[29] H. Klein, "The EPAC architecture: an expert cell approach to field programmable analog devices," *Analog Integrated Circuits and Signal Processing*, vol. 17, no. 1, pp. 91–103, 1998.

[30] H. Kutuk and S. Kang, "A switched capacitor approach to field-programmable analog array (FPAA) design," *Analog Integrated Circuits and Signal Processing*, vol. 17, no. 1, pp. 51–65, 1998.

[31] E. Lee and W. Hui, "A Novel Switched-Capacitor Based Field-Programmable Analog Array Architecture," *Analog Integrated Circuits and Signal Processing*, vol. 17, no. 1-2, pp. 35–50, 1998.

[32] S. Chang, B. Hayes-Gill, and C. Paull, "Multi-function block for a switched current field programmable analogue array," in *IEEE 39th Midwest symposium on Circuits and Systems, 1996.*, vol. 1, 1996.

[33] Anadigm, "Anadigm Designer." `http://www.anadigm.com`; accessed July 16, 2009.

[34] P. Chow and P. Gulak, "A Field-Programmable Mixed-Analog-Digital Array," *Field-Programmable Gate Arrays, 1995. FPGA'95. Proceedings of the Third International ACM Symposium on*, pp. 104–109, 1995.

[35] C. Zhang, A. Bratt, and I. Macbeth, "A New Field Programmable Mixed-Signal Array and Its Applications," *The 4th Canadian Workshop on Field-Programmable Devices*, pp. 132–137, 1996.

[36] R. Edwards, K. Strohbehn, and S. Jaskulek, "A field-programmable mixed-signal array architecture using anti-fuse interconnects," in *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, vol. 3, 2000.

[37] D. D'Mello and P. Gulak, "Design Approaches to Field-Programmable Analog Integrated Circuits," *Analog Integrated Circuits and Signal Processing*, vol. 17, no. 1, pp. 7–34, 1998.

[38] A. Stoica, R. Zebulum, D. Keymeulen, R. Tawel, T. Daud, and A. Thakoor, "Reconfigurable VLSI architectures for evolvable hardware: fromexperimental field programmable transistor arrays to evolution-oriented chips," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 9, no. 1, pp. 227–232, 2001.

[39] J. Luo, J. Bernstein, J. Tuchman, H. Huang, K. Chung, and A. Wilson, "A High Performance Radiation-Hard Field Programmable Analog Array," in *Quality Electronic Design, 2004. Proceedings. 5th International Symposium on*, pp. 522–527, 2004.

[40] J. Becker and Y. Manoli, "A continuous-time field programmable analog array (FPAA) consisting of digitally reconfigurable G/sub M/-cells," in *Circuits and Systems, 2004. ISCAS'04. Proceedings of the 2004 International Symposium on*, vol. 1, 2004.

[41] T. Slaughter and C. Stroud, "Fault injection emulation for field programmable analog arrays," in *Mixed-Signal Design, 2003. Southwest Symposium on*, pp. 212–216, 2003.

[42] T. Balen, A. Andrade Jr, F. Azaïs, M. Lubaszewski, and M. Renovell, "An Approach to the Built-In Self-Test of Field Programmable Analog Arrays," in *VLSI Test Symposium, 2004. Proceedings. 22nd IEEE*, pp. 383–388, 2004.

[43] H. Wang and S. Vrudhula, "Behavioral synthesis of field programmable analog array circuits," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 7, no. 4, pp. 563–604, 2002.

[44] S. Ganesan and R. Vemuri, "Behavioral partitioning in the synthesis of mixed analog-digital systems," *Proc. Design Automation Conference*, pp. 133–138, 2001.

[45] S. Ganesan and R. Vemuri, "Technology mapping and retargeting for field-programmable analog arrays," *Proceedings of the conference on Design, automation and test in Europe*, pp. 58–65, 2000.

[46] S. Ganesan and R. Vemuri, "A methodology for rapid prototyping of analog systems," *International Conference on Computer Design*, pp. 482–488, 1999.

[47] H. Wang, S. Vrudhula, and O. Palusinski, "performance driven placement and routing for field programmable analog arrays," in *Proc. Intl. Conf. on Mixed Design of Integrated Circuits and Systems*, pp. 207–212, 2001.

[48] S. Ganesan and R. Vemuri, "FAAR: A Router for Field-Programmable Analog Arrays," *Proc. Intl. Conf. on VLSI Design*, pp. 556–563, 1999.

[49] J. Cong and Y. Ding, "FlowMap: an optimal technology mapping algorithm for delayoptimization in lookup-table based FPGA designs," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 13, no. 1, pp. 1–12, 1994.

[50] J. Cong and Y. Ding, "Combinational logic synthesis for LUT based field programmable gate arrays," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 1, no. 2, pp. 145–204, 1996.

[51] S. Thakur, Y. Chang, D. Wong, and S. Muthukrishnan, "Algorithms for an FPGA Switch Module Routing Problem with Application to Global Routing," *IEEE TRANSACTIONS ON COMPUTER AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS*, vol. 16, pp. 32–46, 1997.

[52] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," *LECTURE NOTES IN COMPUTER SCIENCE*, pp. 213–222, 1997.

[53] S. Nag and R. Rutenbar, "Performance-driven simultaneous placement and routing for FPGA's," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 17, no. 6, pp. 499–518, 1998.

[54] N. Togawa, M. Yanagisawa, and T. Ohtsuki, "Maple-opt: A Performance-Oriented Simultaneous Technology Mapping, Placement, and Global Routing Algorithm for FPGA's," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 9, pp. 803–818, 1998.

[55] C. Ebeling, L. McMurchie, S. Hauck, and S. Burns, "Placement and routing tools for the Triptych FPGA," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 3, no. 4, pp. 473–482, 1995.

[56] M. Pedram, B. Nobandegani, and B. Preas, "Design and analysis of segmented routing channels for row-basedFPGA's," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 13, no. 12, pp. 1470–1479, 1994.

[57] J. Cong and C. Wu, "An efficient algorithm for performance-optimal FPGA technologymapping with retiming," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 17, no. 9, pp. 738–748, 1998.

[58] J. Cong and Y. Hwang, "Boolean matching for LUT-based logic blocks with applications toarchitecture evaluation and technology mapping," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 20, no. 9, pp. 1077–1090, 2001.

[59] U. Berkeley, "Berkeley logic interchange format (BLIF)," *Oct Tools Distribution*, vol. 2.

[60] A. Vladimirescu, "SPICE-the fourth decade analog and mixed-signal simulation-a stateof the art," in *Semiconductor Conference, 1999. CAS'99 Proceedings. 1999 International*, vol. 1, 1999.

[61] T. Hall, P. Hasler, and D. Anderson, "FieldProgrammable Analog Arrays: AFloating-Gate Approach," *Field-programmable Logic and Applications: Reconfigurable Computing is Going Mainstream: 12th International Conference, FPL 2002, Montpellier, France, September 2-4, 2002: Proceedings*, 2002.

[62] P. Allen and D. Holberg, *CMOS Analog Circuit Design*. Oxford University Press, 2002.

[63] T. Hall, C. Twigg, P. Hasler, and D. Anderson, "Application performance of elements in a floating-gate FPAA," *Circuits and Systems, 2004. ISCAS'04. Proceedings of the 2004 International Symposium on*, vol. 2, 2004.

[64] P. Gray and R. Meyer, *Analysis and Design of Analog Integrated Circuits*. John Wiley & Sons, Inc. New York, NY, USA, 1990.

[65] A. Basu, C. Twigg, S. Brink, P. Hasler, C. Petre, S. Ramakrishnan, S. Koziol, and C. Schlottmann, "RASP 2.8: A New Generation of Floating-gate based Field Programmable Analog Array," in *IEEE Custom Integrated Circuits Conference, 2008. CICC 2008*, pp. 213–216, 2008.

[66] F. Baskaya, S. Reddy, S. Lim, and D. Anderson, "Placement for large-scale floating-gate field-programable analog arrays," *IEEE transactions on very large scale integration(VLSI) systems*, vol. 14, no. 8, pp. 906–910, 2006.

[67] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: applications in VLSI domain," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 7, no. 1, pp. 69–79, 1999.

[68] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[69] L. McMurchie and C. Ebeling, "Wirec 3.2 Tutorial and Reference Manual."

[70] L. McMurchie and C. Ebeling, "PathFinder: a negotiation-based performance-driven router for FPGAs," in *Proceedings of the 1995 ACM third international symposium on Field-programmable gate arrays*, pp. 111–117, ACM New York, NY, USA, 1995.

[71] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, W. Fang, and J. Rose, "VPR 5.0: FPGA cad and architecture exploration tools with single-driver routing, heterogeneity and process scaling," *Field-Programmable Gate Arrays, 2009. Proceedings of the 17th International ACM/SIGDA Symposium on*, pp. 133–142, 2009.

[72] C. Lee, "An algorithm for path connection and its application IRE Trans," *Electronic Computer*, vol. 10, pp. 346–365, 1961.

[73] C. Twigg, J. Gray, and P. Hasler, "Programmable Floating Gate FPAA Switches Are Not Dead Weight," in *Circuits and Systems, 2007, ISCAS 2007 IEEE International Symposium on Circuits and Systems*, pp. 169–172, 2007.

[74] F. Baskaya, B. Gestner, C. Twigg, S. Lim, D. Anderson, and P. Hasler, "Rapid Prototyping of Large-scale Analog Circuits With Field Programmable Analog Array," *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*, pp. 319–320, 2007.

[75] T. Laxminidhi and S. Pavan, "Design Centering High Frequency Integrated Continuous-Time Filters," *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, pp. 1939–1942, 2007.

[76] M. Smith, "WinSpice." `http://www.winspice.co.uk`; accessed July 16, 2009.

[77] B. Sheehan, "TICER: Realizable reduction of extracted RC circuits," in *Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design*, pp. 200–203, IEEE Press Piscataway, NJ, USA, 1999.

[78] B. Mei, A. Lambrechts, J. Mignolet, D. Verkest, and R. Lauwereins, "Architecture exploration for a reconfigurable architecture template," *IEEE Design & Test of Computers*, vol. 22, no. 2, pp. 90–101, 2005.

[79] K. Siozios, A. Bartzas, and D. Soudris, "Architecture-Level Exploration of Alternative Interconnection Schemes Targeting 3D FPGAs: A Software-Supported Methodology," *International Journal of Reconfigurable Computing*, 2008.

[80] K. Siozios, K. Tatas, G. Koutroumpezis, D. Soudris, and A. Thanailakis, "An integrated framework for architecture level exploration of reconfigurable platform," in *Field Programmable Logic and Applications, 2005. International Conference on*, pp. 658–661, 2005.

[81] I. Kuon and J. Rose, "Automated transistor sizing for FPGA architecture exploration," in *Proceedings of the 45th annual conference on Design automation*, pp. 792–795, ACM New York, NY, USA, 2008.

[82] S. Singh, J. Rose, P. Chow, and D. Lewis, "The effect of logic block architecture on FPGA performance," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 3, pp. 281–287, 1992.

[83] J. Rose, R. Francis, D. Lewis, and P. Chow, "Architecture of field-programmable gate arrays: the effect of logicblock functionality on area efficiency," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 5, pp. 1217–1225, 1990.

[84] D. Densmore, A. Donlin, and A. Sangiovanni-Vincentelli, "FPGA architecture characterization for system level performance analysis," in *Proceedings of the conference on Design, automation and test in Europe: Proceedings*, pp. 734–739, European Design and Automation Association 3001 Leuven, Belgium, Belgium, 2006.

[85] P. Maidee and K. Bazargan, "Defect-tolerant FPGA architecture exploration," in *Proceedings of the 13th IEEE Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–6, 2006.

[86] H. Parvez, Z. Marrakchi, U. Farooq, and H. Mehrez, "A New Coarse-Grained FPGA Architecture Exploration Environment," in *Field-Programmable Technology, 2008. FPT 2008. International Conference on*, pp. 285–288, 2008.

[87] A. Chattopadhyay, W. Ahmed, K. Karuri, D. Kammler, R. Leupers, G. Ascheid, and H. Meyr, "Design space exploration of partially re-configurable embedded processors," in *Proceedings of the conference on Design, automation and test in Europe*, pp. 319–324, EDA Consortium San Jose, CA, USA, 2007.

[88] H. Zhang, M. Wan, V. George, and J. Rabaey, "Interconnect architecture exploration for low-energy reconfigurable single-chip DSPs," in *Proceedings IEEE Computer Society Workshop On VLSI'99*, pp. 2–8, 1999.

[89] V. Nookala, Y. Chen, D. Lilja, and S. Sapatnekar, "Microarchitecture-aware floorplanning using a statistical design of experiments approach," in *Proceedings of the 42nd annual conference on Design automation*, pp. 579–584, ACM New York, NY, USA, 2005.

[90] Y. Wu, D. Chang, M. Marek-Sadowska, and S. Tsukiyama, "Not necessarily more switches more routability," *Proc. ASP-DAC*, pp. 579–584, 1997.

[91] R. Fisher, "Design of experiments," *British Medical Journal*, vol. 1, no. 3923, p. 554, 1936.

[92] V. Czitrom and P. Spagon, *Statistical case studies for industrial process improvement*. Society for Industrial Mathematics, 1997.

[93] M. McKay, R. Beckman, and W. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, pp. 55–61, 2000.

[94] D. Raj, *Sampling theory*. McGraw-Hill, New York, 1968.

[95] E. Harrington, "The desirability function," *Industrial Quality Control*, vol. 21, no. 10, pp. 494–498, 1965.

[96] G. Derringer and R. Suich, "Simultaneous optimization of several response variables," *Journal of Quality Technology*, vol. 12, no. 4, pp. 214–219, 1980.

[97] V. Torczon, "On the convergence of pattern search algorithms," *SIAM J. Optim.*, vol. 7, no. 1, pp. 1–25, 1997.

[98] V. Torczon and M. Trosset, "From evolutionary operation to parallel direct search: Pattern search algorithms for numerical optimization," *Computing Science and Statistics*, pp. 396–401, 1998.